UM OTIMIZADOR DINÂMICO PARA A MATERIALIZAÇÃO DE DOCUMENTOS XML ATIVOS

Gabriela Gouveia Guedes Loureiro Ruberg

TESE SUBMETIDA AO CORPO DOCENTE DA COORDENAÇÃO DOS PROGRAMAS DE PÓS-GRADUAÇÃO DE ENGENHARIA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE DOUTOR EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

Aprovada por:

Profª. Marta Lima de Queirós Mattoso, D.Sc.

Prof. Jano Moreira de Souza, Ph.D.

Prof. Nelson Francisco Favilla Ebecken, D.Sc.

Prof. Marco Antônio Casanova, Ph.D.

Prof. Carlos Alberto Heuser, Doutor

RIO DE JANEIRO, RJ – BRASIL

OUTUBRO DE 2007

RUBERG, GABRIELA GOUVEIA GUEDES LOUREIRO

Um Otimizador Dinâmico para a Materialização de Documentos XML Ativos [Rio de Janeiro] 2007

XVI, 231 p. 29,7 cm (COPPE/UFRJ, D.Sc., Engenharia de Sistemas e Computação, 2007)

Tese – Universidade Federal do Rio de Janeiro, COPPE

1. XML
2. Serviços Web
3. Sistemas Ponto-a-Ponto
4. Otimização Dinâmica

I. COPPE/UFRJ II. Título (série)

Resumo da Tese apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Doutor em Ciências (D.Sc.)

UM OTIMIZADOR DINÂMICO PARA A MATERIALIZAÇÃO DE DOCUMENTOS XML ATIVOS

Gabriela Gouveia Guedes Loureiro Ruberg

Outubro/2007

Orientador: Marta Lima de Queirós Mattoso

Programa: Engenharia de Sistemas e Computação

Da combinação da linguagem XML e dos serviços Web, surgiu uma nova classe de documentos ditos XML ativos (ou AXML, de "*Active* XML"), os quais possuem elementos que correspondem a chamadas de serviços Web. Para materializar o conteúdo completo de um documento AXML, é necessário executar todas as suas chamadas de serviços Web, cuja otimização constitui um problema difícil. Técnicas atuais para agendamento da execução de tarefas de *workflows* e processamento distribuído de consultas são insuficientes para resolver esse problema, pois na materialização AXML: (*i*) a especificação completa de um documento AXML não é necessariamente conhecida *a priori*; (*ii*) o resultado de uma chamada de serviço Web pode conter outras chamadas de serviços, exigindo re-planejamento; e (*iii*) devido à volatilidade dos sítios na rede, um plano gerado pelo otimizador pode se tornar inválido no momento de sua execução. A grande maioria dos otimizadores existentes é baseada em coordenação centralizada.

Esta tese propõe uma estratégia de otimização dinâmica baseada em custos para a materialização eficiente de documentos AXML, considerando a volatilidade de um cenário P2P. A estratégia permite que o otimizador reduza o tamanho do espaço de busca, obtenha informações atualizadas sobre os participantes do sistema e construa resultados parciais incrementalmente. A estratégia é capaz de tratar documentos AXML complexos e explora descentralização em vários níveis. Esta tese apresenta ainda uma arquitetura orientada a serviços, chamada **XCraft**, para apoiar as técnicas de otimização propostas. Por intermédio de um protótipo do XCraft, foram obtidos resultados que mostram ganhos significativos de desempenho em relação às estratégias tradicionais (*i.e.*, estáticas e centralizadas).

Abstract of Thesis presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Doctor of Science (D.Sc.)

A DYNAMIC OPTIMIZER FOR THE MATERIALIZATION OF ACTIVE XML DOCUMENTS

Gabriela Gouveia Guedes Loureiro Ruberg

October/2007

Advisor: Marta Lima de Queirós Mattoso

Department: Computer Science and Systems Engineering

An active XML (AXML) document contains special tags that represent calls to Web services. Retrieving its contents consists in materializing its data elements by invoking all its embedded service calls in a P2P network. This implies many equivalent materialization alternatives, with different performance, which represents a hard optimization problem. Current techniques for workflow scheduling and distributed query processing are insufficient for this problem, since in AXML materialization: (*i*) the set of participating peers is not necessarily known in advance; (*ii*) service calls in the result of other calls forbid a simple "optimize-then-execute" strategy; and (*iii*) due to the peer volatility in the network, a plan computed by the optimizer may become invalid at the moment of its execution. Moreover, most of the current optimizers are based on centralized coordination.

We propose a dynamic, cost-based optimization strategy to efficiently materialize AXML documents considering the volatility of a P2P scenario. This strategy enables the optimizer to reduce the size of the search space, get more up-to-date information on the status of the peers, and deliver partial results in advance. The strategy can handle arbitrarily complex AXML documents, and exploits decentralization in many levels. Furthermore, we propose a service-oriented optimization architecture, called **XCraft**, to support the proposed techniques. We evaluated our approach in an XCraft prototype for the *ActiveXML* system, an open-source P2P platform. Empirical results show important performance gains compared to centralized, static materialization strategies.

Aos meus amados pais, Adelma e Edson Loureiro,
com toda a minha admiração.

"Mas é preciso ter força, é preciso ter raça, é preciso ter gana sempre...
Quem traz na pele essa marca possui a estranha mania de ter fé na vida."

Milton Nascimento

"Um coração para sonhar, inquieto e sempre a bater,

ansioso por entender as coisas que Tu disseste.

Eis o que eu venho Te dar, eis o que eu ponho no altar:

Toma, Senhor, que ele é Teu!

Meu coração não é meu."

Padre Zezinho

AGRADECIMENTOS

"Dê-me um ponto de apoio que eu consigo erguer o mundo."
Arquimedes

À querida Profª Marta Mattoso, por tudo. Certamente não há como agradecer suficientemente pelo que aprendi com ela desde os tempos de mestrado. Sua orientação criteriosa e exemplar teve um impacto profundo em mim e na minha vida. Sinto um enorme orgulho em dizer que sou sua aluna, porque sei que isso evoca a excelência com que desenvolve seus projetos de pesquisa científica. Sua imensa dedicação nos habitua ao gosto pela auto-superação e cria grandes oportunidades. Por tudo isso, ser-lhe-ei para sempre grata.

Ao Prof. Serge Abiteboul, idealizador do modelo de documentos AXML, pela excelente acolhida no grupo Gemo durante meu estágio de doutorado sanduíche. Seu profundo e amplo conhecimento sobre Computação ajudou bastante a solidificar as estruturas do meu trabalho. Com ele, pude entender claramente que o ócio criativo é uma ferramenta essencial para o sucesso e que embora o desenvolvimento de pesquisa científica seja árduo, ele deve ser sempre divertido. Merci beaucoup, Serge!

À Profª Ioana Manolescu, pela importante colaboração na análise de desempenho da materialização AXML. Ao pessoal do projeto Active XML, cuja plataforma P2P eu utilizei para construir o protótipo do otimizador XCraft. A Nida Azis e a Benjamin Nguyen, pelas trocas de idéias no desenvolvimento do sistema Acware. Ao Prof. Bernd Aman e a Radu Pop pelas discussões enriquecedoras sobre o uso de referências abstratas para serviços Web em documentos AXML.

Aos professores Carlos Alberto Heuser, Marco Antônio Casanova, Nelson Ebecken e Jano Souza, pela participação na banca examinadora desta tese e pelos comentários generosos.

Durante o doutorado, eu tive algumas oportunidades de discutir o meu trabalho com professores do PESC, cujos comentários foram fundamentais para a evolução da

tese. Em particular, eu agradeço as contribuições dos professores Gerson Zaverucha, Inês Castro e Cláudia Werner.

Ao pessoal administrativo do PESC, por todo apoio operacional recebido e pelo constante incentivo.

À amiga Daniela Marques Pereira, com quem compartilhei meses de trabalho sobre a otimização da materialização AXML. Sua dedicação, vontade de aprender e eficiência permitiram o desenvolvimento do ambiente de simulação SiMAX, que foi muito importante para avaliar o funcionamento do otimizador XCraft em diferentes cenários. Além disso, a alegria e o espírito positivo de Dani tornaram esse período inesquecível.

A Alexandre Silva, pelas inúmeras discussões sobre o modelo de fragmentação de dados do PartiX e pela obtenção dos resultados experimentais que foram fundamentais para avaliar as técnicas propostas. Aos colegas Humberto Vieira, Valdino Azevedo e Cláudio Ferraz, pela colaboração em trabalhos desenvolvidos durante o doutorado.

Ao Banco Central do Brasil, pela licença remunerada que permitiu o desenvolvimento desta tese de doutorado, por intermédio do Programa de Pós-Graduação do Sistema Banco Central de Educação Permanente. À chefia do Departamento de Operações do Mercado Aberto (Demab) e da Divisão de Administração do Selic (Dicel), especialmente a Ivan Luís Gonçalves de Oliveira Lima, Ruben de Almeida Galvão e Marcus Antônio Sucupira, por todo apoio recebido tanto na aprovação do afastamento para o doutorado como no fechamento desta tese. Ao meu orientador técnico, José Félix Furtado de Mendonça, pela atenção e pelo pronto encaminhamento de todas as minhas demandas.

Ao meu coordenador e querido amigo José Roberto Brasileiro de Siqueira (grande Zero!), por absolutamente tudo. Espero poder demonstrar e retribuir sempre minha profunda gratidão pelo seu apoio. Aos colegas da equipe Integração, pela torcida e por segurar as pontas nas minhas ausências: André Castanheira, Guilherme Figueiredo e Rafael Barros.

A Ângela Musiello, por sua amizade e orientações essenciais para a concretização deste trabalho.

Ao Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq), pela bolsa de doutorado sanduíche que me permitiu realizar o estágio de um ano na França.

Ao meu amado marido, Nicolaas Ruberg, que é a pedra fundamental de minha existência e das minhas conquistas. Nem nos meus sonhos mais criativos eu imaginei encontrar alguém tão especial na minha vida. Que Deus prolongue pela eternidade a felicidade de tê-lo ao meu lado.

A toda a minha família, pelo amor, carinho e fé nesta jornada.

Pela torcida e encorajamento, aos amigos: Micheline e Antônio Silveira, Valmir Vaz, Gláucia e Gustavo Pinto, Cláudia e Alexandre Lima, Robson Pinheiro, Renata Guedes Pereira, Vanessa e Leonardo Murta, Pablo Florentino e Jonice Oliveira. À querida galera da Dataprev: Ana Caldas, Lia, Cristina, Renato, Patrícia e Beni.

To the people I met in France, for the great time we had together: Nicoleta Preda, Bogdan Cautis, Cristina Sirângelo, Zoë Abrams, Stéphanie and Alexander, Prof. Tova Milo, Prof. Susan Davidson, and Prof. Victor Vianu.

ÍNDICE DO TEXTO

AGRADECIMENTOS VII

ÍNDICE DE FIGURAS XIV

ÍNDICE DE TABELAS XVI

CAPÍTULO 1. INTRODUÇÃO 1

1.1 MOTIVAÇÃO 2

1.2 CARACTERIZAÇÃO DO PROBLEMA 4

1.3 OBJETIVOS 7

1.4 ORGANIZAÇÃO DO TEXTO 12

CAPÍTULO 2. FUNDAMENTOS DE DOCUMENTOS AXML 13

2.1 INTRODUÇÃO 14

2.2 CASO DE USO: CONTROLE DE *SWAP* CAMBIAL 16

2.2.1 Documento Ativo SwapWorkspace 17

2.2.2 Alternativas de Materialização AXML 21

2.2.3 Oportunidades de Otimização 24

2.3 MODELO DE DOCUMENTOS AXML 26

2.3.1 Conteúdo de um Documento AXML 28

2.3.2 Chamadas de Serviços Web 29

2.3.3 Parâmetros de Entrada definidos por Consultas 32

2.3.4 Chamadas Colaterais 32

2.4 PLATAFORMA *ACTIVEXML* 33

2.4.1 Arquitetura de Sistema 34

2.4.2 Propostas de Otimização da Materialização AXML 36

CAPÍTULO 3. *WORKFLOWS* DISTRIBUÍDOS: ESTADO DA ARTE 41

3.1 INTRODUÇÃO 42

3.2 REPRESENTAÇÃO DE *WORKFLOWS* COMPLEXOS 45

3.2.1 Padrões de Fluxos de Controle e de Dados 46

3.2.2 *Resultados Persistentes* ... 48
3.2.3 *Evolução Dinâmica* ... 48
3.3 PLANEJAMENTO DA EXECUÇÃO DE *WORKFLOWS* ... 51
3.3.1 *Uso de Heurísticas* ... 52
3.3.2 *Classificação dos Otimizadores* ... 54
3.4 OTIMIZADORES DESCENTRALIZADOS ... 59
3.4.1 *MENTOR* ... 60
3.4.2 *ADEPT-WfMS* ... 61
3.4.3 *GridFlow* ... 63
3.4.4 *Triana* ... 64
3.4.5 *D2WMS* ... 65
3.4.6 *GRAND* ... 66
3.4.7 *Self-Serv* ... 68
3.4.8 *ASKALON* ... 69
3.4.9 *Comentários Gerais* ... 70
3.5 ANÁLISE DE DESEMPENHO DE SERVIÇOS WEB ... 71
3.6 CONSIDERAÇÕES FINAIS ... 74

CAPÍTULO 4. UM MODELO CANÔNICO PARA A MATERIALIZAÇÃO DE DOCUMENTOS AXML ... 75

4.1 INTRODUÇÃO ... 76
4.2 DEPENDÊNCIAS DE INVOCAÇÃO ... 77
4.2.1 *Parâmetros Intencionais Concretos* ... 78
4.2.2 *Parâmetros Intencionais Não-Concretos* ... 79
4.2.3 *Transitividade* ... 81
4.3 CHAMADAS COLATERAIS ... 82
4.4 GRAFO DE DEPENDÊNCIAS ... 83
4.4.1 *Estados de uma Chamada de Serviço Web* ... 84
4.4.2 *Definição Preliminar do Grafo* ... 85
4.4.3 *Condições de Validade do Grafo* ... 86
4.4.4 *Eliminação de Redundâncias* ... 89
4.4.5 *Pontos de Saída* ... 91
4.5 ATUALIZAÇÃO DINÂMICA DO GRAFO ... 91
4.5.1 *Algoritmo Básico de Atualização* ... 93

4.5.2 *Atualização baseada em Nodos Pipe* ... 96
4.5.3 *Validade do Grafo Atualizado* ... 97
4.5.4 *Controle das Atualizações do Grafo* ... 98
4.5.5 *Comentários* ... 102
4.6 MODELO P2P PARA MATERIALIZAÇÃO AXML ... 103
4.6.1 *Especificações Abstratas de Documentos AXML* ... 103
4.6.2 *Uso de Serviços Web Equivalentes* ... 104
4.6.3 *Alternativas de Colaboração P2P* ... 106
4.6.4 *Instâncias da Materialização AXML* ... 110
4.7 CONSIDERAÇÕES FINAIS ... 112

CAPÍTULO 5. UMA ESTRATÉGIA DINÂMICA E DESCENTRALIZADA PARA A MATERIALIZAÇÃO DE DOCUMENTOS AXML ... 113
5.1 INTRODUÇÃO ... 114
5.2 PLANEJAMENTO DA MATERIALIZAÇÃO AXML ... 116
5.2.1 *Espaço de Planos Equivalentes* ... 118
5.2.2 *Métrica de Otimização* ... 119
5.2.3 *Principais Topologias do Problema* ... 121
5.3 ETAPAS DA ESTRATÉGIA DE OTIMIZAÇÃO ... 123
5.4 GERAÇÃO DE PLANOS ABSTRATOS ... 126
5.4.1 *Árvores Geradoras* ... 126
5.4.2 *Álgebra de Operadores para Materialização AXML* ... 128
5.4.3 *Plano Abstrato Inicial* ... 130
5.5 GERAÇÃO DE SUBPLANOS FÍSICOS ... 132
5.5.1 *Identificação de Pontos de Quebra* ... 132
5.5.2 *Agendamento da Avaliação de Subplanos* ... 133
5.5.3 *Enumeração das Alternativas de Materialização* ... 135
5.5.4 *Execução de Subplanos Físicos* ... 137
5.6 COMENTÁRIOS ... 138

CAPÍTULO 6. ANÁLISE DAS TÉCNICAS PROPOSTAS ... 140
6.1 INTRODUÇÃO ... 141
6.2 AMBIENTE DE TESTE ... 141
6.2.1 *Protótipo do XCraft* ... 142

6.2.2 Rede Ponto-a-Ponto *142*
6.2.3 Geração de Documentos AXML *143*
6.2.4 Serviços Web *144*
6.3 Análise do Espaço de Busca 145
6.3.1 Variação do Número de Sítios *146*
6.3.2 Variação do fanIn das Chamadas de Serviços Web *148*
6.3.3 Geração Dinâmica de Planos de Materialização *150*
6.4 Efeitos da Delegação de Chamadas de Serviços 152
6.5 Geração de Planos por meio de Busca Local 153
6.6 Fragmentação de Documentos XML 154

Capítulo 7. Conclusões e Trabalhos Futuros 156
7.1 Considerações Finais 157
7.2 Contribuições 160
7.3 Trabalhos Futuros 164

Referências Bibliográficas 167

Anexo I – Relatório Técnico 185

ÍNDICE DE FIGURAS

Figura 1. Diagrama de caso de uso da aplicação *CurrencySwap*. 17
Figura 2. Documento AXML *SwapWorkspace*. 19
Figura 3. Versão parcialmente materializada do documento ativo *SwapWorkspace* após a invocação de sc5. 20
Figura 4. Cenário P2P da aplicação CurrencySwap. 21
Figura 5. Materialização AXML por meio de avaliação centralizada. 23
Figura 6. Algumas alternativas descentralizadas para materialização AXML. 23
Figura 7. Componentes de um sítio *ActiveXML* 34
Figura 8. Principais características de otimizadores para a execução de *workflows* distribuídos, baseadas na taxonomia de YU e BUYYA (2005). 54
Figura 9. Planejamento dinâmico "*just-in-time*" no otimizador Pegasus (DEELMAN *et al.*, 2004). 56
Figura 10. Arquitetura do sistema GRAND (VARGAS *et al.*, 2005). 67
Figura 11. Diagrama de estados de chamadas de serviços Web. 84
Figura 12. Grafo de dependências do documento *SwapWorkspace*. 86
Figura 13. Padrões básicos de ciclos maliciosos em um documento AXML. 87
Figura 14. Grafo de dependências sem redundâncias. 89
Figura 15. Atualização de um grafo de dependências por uma resposta ativa. 95
Figura 16. Atualização do grafo de dependências com nodos *pipe* 96
Figura 17. Delegação de subplanos na materialização de documentos AXML. 107
Figura 18. Passos da estratégia dinâmica e descentralizada para a otimização da materialização de documentos AXML. 124
Figura 19. Árvores geradoras do documento *SwapWorkspace*. 128
Figura 20. Plano abstrato inicial gerado para o documento *SwapWorkspace*. 131
Figura 21. Pontos de quebra da tarefa de materialização. 133
Figura 22. Guia-S para ordenar a avaliação dos subplanos da tarefa de μ(sc5). 134
Figura 23. Alguns subplanos físicos equivalentes gerados para o operador μ(sc7). 136
Figura 24. Passos da avaliação de cada operador local de um subplano físico. 137
Figura 25. Sistema P2P utilizado nos testes experimentais reais. 142
Figura 26. Tipos de grafos de dependências dos testes experimentais. 144

Figura 27. Serviço Web declarativo que retorna um documento com 50K*bytes*. 144
Figura 28. Crescimento do espaço de busca com a variação do número de sítios, considerando $|\Lambda|=4$ e *fanIn*=1. 146
Figura 29. Comparação de estratégias na variação do número de sítios. 147
Figura 30. Crescimento do espaço de busca com a variação do número de parâmetros intencionais das chamadas de serviços, considerando *#sítios*=3 e $|\Lambda|=4$. 148
Figura 31. Comparação de estratégias na variação do *fanIn*. 149
Figura 32. Número de planos gerados em diferentes estratégias de otimização para h=8, *fanIn*=1, *#sítios*=3 e $|\Lambda|=4$, variando o fator de quebra. 150
Figura 33. Tempo de otimização de diferentes estratégias de otimização. 151
Figura 34. Ganho de desempenho obtido pela delegação de tarefas no XCraft, em relação à abordagem centralizada. 152
Figura 35. Resultados da estratégia SLS-MC com diferentes heurísticas de agendamento de invocações. 154

ÍNDICE DE TABELAS

Tabela 1. Distribuição de serviços Web da aplicação *CurrencySwap*. 22
Tabela 2. Padrões de fluxos de controle encontrados em documentos AXML..... 46
Tabela 3. Serviços para colaboração P2P na materialização de documentos AXML.. 109
Tabela 4. Principais topologias de grafos de dependências..... 122
Tabela 5. Álgebra de operadores para a otimização dinâmica e descentralizada da materialização de documentos AXML. 129
Tabela 6. Configuração dos sítios usados em testes experimentais reais..... 143

Capítulo 1

INTRODUÇÃO

Este capítulo apresenta a motivação, o contexto e os principais objetivos desta tese, além de um resumo sobre os capítulos restantes e a organização do texto.

"Wholly new forms of encyclopedias will appear, ready made with a mesh of associative trails running through them, ready to be dropped into the memex and there amplified."

As We May Think (BUSH, 1945)

1.1 MOTIVAÇÃO

A crescente necessidade de integração e colaboração das sociedades modernas impulsionou avanços tecnológicos significativos na gerência de informações distribuídas, especialmente nas aplicações da *Web* (ABITEBOUL *et al.*, 2005b). Em um mundo cada vez mais interconectado e globalizado, o alcance no uso de recursos por pessoas e instituições tem ultrapassado as fronteiras tradicionais, formando arranjos complexos de sistemas computacionais. Muitas vezes, tais arranjos ocorrem em cenários dinâmicos com configurações *ad-hoc*. Assim, espera-se que esses sistemas ofereçam funcionalidades personalizadas, capazes de se adaptar tanto a preferências individuais como a diferentes cenários de utilização. Além disso, a complexidade decorrente da distribuição e heterogeneidade dos recursos nestes cenários evidencia a importância de mecanismos que permitam criar níveis de abstração sobre a infra-estrutura computacional. Para serem efetivamente utilizados, esses mecanismos devem apresentar desempenho eficiente, baixo potencial de falhas e alta capacidade de recuperação.

Os sistemas baseados em comunicação ponto-a-ponto (MILOJICIC *et al.*, 2002), amplamente conhecidos como sistemas P2P (de "*Peer to Peer*", em Inglês), surgiram como uma alternativa promissora para atender os requisitos de ambientes colaborativos para integração de informações e compartilhamento de recursos. Consolidando essa tendência, técnicas para gerência de dados em sistemas P2P têm sido vastamente exploradas nos últimos anos (ABITEBOUL, 2003, ABITEBOUL *et al.*, 2002, 2004b, 2005c, BRAGA, 2005, COOPER e GARCIA-MOLINA, 2005, CRESPO e GARCIA-MOLINA, 2002, 2004, GANESAN e GARCIA-MOLINA, 2005, HALEVY *et al.*, 2003, HUEBSCH *et al.*, 2005, MILO *et al.*, 2003, NG *et al.*, 2003, NEJDL *et al.*, 2003, PAPADIMOS e MAIER, 2002, PAPADIMOS *et al.*, 2003, PITOURA *et al.*, 20003, RUBERG *et al.*, 2003, SARTIANI *et al.*, 2004, SCHULER *et al.*, 2004, STOICA *et al.*,

2001, TATARINOV *et al.*, 2003). Neste contexto, duas tecnologias representam papéis cruciais: o formato XML (XML, 2006), como a *lingua franca* para troca de dados; e os serviços Web (WEB SERVICES, 2006), como o padrão para integração de dados e interoperabilidade de programas heterogêneos.

Serviços Web são geralmente descritos como programas independentes e autocontidos, cuja interface pode ser publicada, descoberta e acessada por intermédio da Web (PIRES, 2002, KREGER, 2001). Eles estão associados a uma pilha basilar de padrões, que são: o protocolo de mensagens SOAP (*Simple Object Access Protocol*, SOAP, 2003); a linguagem WSDL (*Web Services Description Language*, WSDL, 2006) para descrição de serviços; e o protocolo UDDI (*Universal Description; Discovery, and Integration*, UDDI, 2004) para a publicação e descoberta de serviços. Da combinação da linguagem XML e da tecnologia de serviços Web, surgiram modelos interessantes para expressar computações distribuídas. Por exemplo, por meio de vocabulários XML próprios, linguagens como a *Apache Jelly* (JELLY, 2006) e a *Microsoft* XAML (GRIFFITHS, 2004) permitem que chamadas de serviços Web sejam declaradas dentro de documentos XML. Em um contexto P2P, essa combinação resultou em uma nova classe de documentos, ditos documentos XML ativos (ou, simplesmente, documentos AXML – de "*Active XML*"), os quais consistem em uma mídia altamente adaptativa para representar informações distribuídas (ABITEBOUL *et al.*, 2003a, 2003b, 2004c, ABITEBOUL, NGUYEN e RUBERG, 2006b, MILO *et al.*, 2003, RUBERG *et al.*, 2004a, PEREIRA, RUBERG e MATTOSO, 2006, TAROPA *et al.*, 2005).

Documentos AXML foram inicialmente definidos no contexto da plataforma *ActiveXML* (ACTIVEXML, 2002, ABITEBOUL *et al.*, 2002, 2004c, 2005c), um sistema P2P de *software* livre que visa prover mecanismos para a gerência declarativa de dados e programas na Web. Um documento AXML contém elementos especiais que representam chamadas de serviços Web. Tais chamadas podem ser executadas automaticamente ou sob demanda, tendo como conseqüência a atualização do conteúdo do documento com o resultado da respectiva execução. As chamadas de serviços Web embutidas em um documento AXML podem ser vistas como elementos intencionais. Ou seja, é necessário materializá-los para se recupere o conteúdo que eles representam. Observe que, neste ponto, documentos AXML são semelhantes a visões: para recuperar o conteúdo completo de um documento AXML, deve-se materializar todas as suas

chamadas de serviços Web (*i.e.*, invocando-as e inserindo seus resultados no documento). Vale ressaltar que esse processo de materialização é bastante comum em aplicações da Web, onde o acesso ao conteúdo de um documento geralmente requer a reconstrução dos dados que o compõem. Contudo, devido à heterogeneidade e ao dinamismo dos sistemas P2P, materializar eficientemente documentos AXML constitui um problema de otimização bastante complexo.

1.2 CARACTERIZAÇÃO DO PROBLEMA

A materialização de um documento AXML é semelhante à execução de um *workflow* (HOLLINGSWORTH, 1995, WFMC, 1999, YU e BUYYA, 2005) cujas tarefas são chamadas de serviços Web. Como essas chamadas costumam estar inter-relacionadas, durante o processo de materialização deve ser respeitada uma determinada ordem de invocação. Por exemplo, pelo aninhamento de elementos intencionais podem ocorrer dependências de invocação. Ou seja, quando algumas chamadas de serviços Web utilizam o resultado de outras chamadas como parâmetros de entrada. Neste caso, além da ordem de invocação, também é importante identificar os resultados de chamadas de serviços que são realmente necessários para compor o conteúdo do documento AXML. Alguns resultados são apenas intermediários e não precisam ser armazenados até o final do processo de materialização.

As restrições de invocação de um documento AXML determinam uma ordem parcial na execução das chamadas de serviços Web e correspondem a padrões básicos para controle de *workflows*. Em particular, documentos AXML podem conter seqüências, divisões paralelas, sincronizações e múltiplas instâncias sem sincronização (AALST *et al.*, 2003). Entretanto, a materialização de um documento AXML sempre envolve a transferência de resultados de chamadas de serviços para o sítio que está construindo o conteúdo do documento, dito sítio mestre. Deste modo, o documento pode ser composto (e, conseqüentemente, consumido) incrementalmente. Já em sistemas para a execução de *workflows* distribuídos, como o Globus (GLOBUS, 2006, FOSTER, 2005), o Taverna (TAVERNA, 2006, OINN *et al.*, 2004) e o Triana (TRIANA, 2003, TAYLOR *et al.*, 2003, MAJITHIA *et al.*, 2004), resultados parciais são raramente úteis. Observe que geralmente esses sistemas são baseados em grades computacionais (mais conhecidas como *grids*) (FOSTER e KESSELMAN, 2000).

Um aspecto crítico da materialização de documentos AXML decorre da possibilidade de respostas ativas. Em sistemas P2P baseados nesses documentos, o resultado da execução de uma chamada de serviço Web pode conter outras chamadas de serviços. Isto significa que a especificação do problema (*i.e.*, o documento AXML a ser materializado) pode mudar em tempo de execução, demandando a atualização do plano de materialização e possivelmente invalidando decisões de otimização anteriores. Este problema é bem diferente do que ocorre em cenários tradicionais, como os de otimização de consultas distribuídas. Uma consulta pode ser otimizada seja estaticamente ou dinamicamente (BOUGANIM *et al.*, 2000, COLE e GRAEFE, 1994, ELMASRI e NAVATHE, 1994), mas a especificação da consulta em si não é alterada em tempo de execução. Mesmo para consultas parametrizadas, tipicamente encontradas em bancos de dados relacionais (IOANNIDIS *et al.*, 1992), apesar de o plano de consulta se modificar conforme os valores de seus parâmetros, as relações envolvidas são sempre as mesmas. Ao lidar com documentos AXML, o sistema deve ser capaz de atualizar planos de materialização dinamicamente para refletir as alterações decorrentes de respostas ativas. Convém que o escopo do impacto dessas mudanças no planejamento seja reduzido, evitando re-otimizações excessivas.

Basicamente, os elementos intencionais de um documento AXML apontam para endereços completos de serviços Web (ABITEBOUL *et al.*, 2002, 2003a, 2003b, 2004b, MILO *et al.*, 2003), incluindo uma URL e outros parâmetros necessários para a execução do serviço (conforme é definido nos padrões SOAP e WSDL). Em uma abordagem mais flexível, chamadas de serviços Web podem conter referências abstratas, baseadas em uma ontologia de serviços Web (OWLS, 2004, MAXIMILIEN e SINGH, 2004). Esta abordagem é bem conveniente para especificar documentos AXML, pois localizar os melhores recursos para executar uma requisição de serviço Web em um sistema P2P costuma ser penoso para os usuários.

Considerando referências abstratas de serviços Web, um documento AXML pode ser materializado por diferentes estratégias de avaliação. Como evidenciado em RUBERG *et al.* (2004a) por meio de uma análise empírica, o desempenho pode variar bastante em cada estratégia adotada. Independente da ordem de invocação das chamadas de serviços, basicamente essas estratégias diferem na escolha: (*i*) do sítio que executa cada chamada de serviço, dentre os provedores dos serviços Web requisitados; e (*ii*) do

sítio que invoca cada chamada de serviço, dentre aqueles que podem colaborar com o sítio mestre. Observe que o sítio mestre pode delegar a outros sítios a invocação de uma chamada de serviço Web mesmo que ela aponte para um endereço específico. Isto permite que a materialização de um documento AXML seja descentralizada, em uma abordagem mais aderente aos sistemas P2P típicos. Essa descentralização pode reduzir significativamente os custos de comunicação das chamadas de serviços Web, pois permite explorar a proximidade de sítios interligados por canais de alta velocidade. Isso evita transferências desnecessárias de resultados intermediários para o sítio mestre.

O problema tratado nesta tese consiste essencialmente em determinar um mapeamento eficiente para a execução de um *workflow* de tarefas inter-relacionadas em um sistema P2P. Não há dúvidas sobre a importância do planejamento na execução de *workflows* em ambientes distribuídos heterogêneos, como em sistemas P2P e *grids*. Todavia, apesar da ampla atenção na literatura (BLYTHE *et al.*, 2003, 2005, BRAUN *et al.*, 1998, 2001, DONG e AKL, 2006, KWOK e AHMAD, 1999, YU e BUYYA, 2005), as abordagens de otimização atuais são geralmente centralizadas (ALHUSAINI *et al.*, 1999, BLYTHE *et al.*, 2004, 2005, DASILVA *et al.*, 2003, DEELMAN *et al.*, 2003, 2004, 2005, FOSTER *et al.*, 2002, GOUNARIS *et al.*, 2004b, HUANG *et al.*, 2005, OINN *et al.*, 2004, PEGASUS, 2006, SAKELLARIOU e ZHAO, 2004a, SINGH *et al.*, 2006, VERMA *et al.*, 2005, WIECZOREK *et al.*, 2005, ZENG *et al.*, 2004), com raras exceções baseadas em redes hierárquicas (CAO *et al.*, 2003, THAIN *et al.*, 2005, VARGAS *et al.*, 2005), sistemas distribuídos (BAUER e DADAM, 2000, MUTH *et al.*, 1998) e redes P2P (ARORA *et al.*, 2002, BENATALLAH *et al.*, 2005).

Além da complexidade inerente dos ambientes distribuídos heterogêneos, a otimização da materialização de documentos AXML tem que lidar com o dinamismo comum dos cenários P2P. Esse dinamismo acontece tanto no desempenho como na participação dos sítios, os quais podem entrar ou sair do sistema aleatoriamente (MILOJICIC *et al.*, 2002). Portanto, é interessante gastar pouco tempo gerando um plano de materialização, já que ele pode se tornar inválido no momento da execução. Ainda porque, devido ao grande número de variáveis a serem consideradas e à natureza combinatória do problema, a geração de um plano ótimo é inviável na maioria dos casos. De fato, imprevisibilidade é geralmente endêmica em cenários de larga escala (KAMMER *et al.*, 2000), como acontece em sistemas P2P e em *grids*. Logo, em vez de

produzir planos completos antes do início da materialização de um documento AXML, é mais interessante que planos parciais sejam gerados e executados incrementalmente, possivelmente em paralelo.

1.3 OBJETIVOS

Esta tese visa apresentar mecanismos para a materialização eficiente de documentos AXML em sistemas P2P. Devido à volatilidade desses sistemas, tornou-se evidente no decorrer deste trabalho que técnicas dinâmicas e descentralizadas são vitais para atingir esse propósito. Assim, esta tese propõe uma estratégia de otimização baseada em custos que analisa a invocação de chamadas de serviços Web considerando as flutuações de um ambiente P2P e explorando a colaboração entre sítios.

Para a elaboração da estratégia proposta, são necessários: (*i*) um modelo canônico para representar as restrições de invocação das chamadas de serviços embutidas em um documento AXML; (*ii*) um algoritmo dinâmico para distribuir corretamente a materialização de documentos AXML, considerando diferentes relacionamentos entre chamadas de serviços e um sistema P2P; (*iii*) uma álgebra de operadores para codificar planos de materialização, especialmente projetada para gerar planos físicos incrementalmente e permitir otimização colaborativa em um ambiente P2P; (*iv*) um modelo de custo para análise de planos equivalentes, com métricas de desempenho adequadas para a invocação de serviços Web e a delegação de tarefas; e (*v*) uma arquitetura de otimização orientada a serviços, com flexibilidade para acomodar diferentes estratégias de materialização de documentos AXML e descentralização.

Embora documentos AXML sejam tradicionalmente representados por árvores (ABITEBOUL *et al.*, 2003a, 2003b, 2004d, MILO *et al.*, 2003), essas estruturas não são capazes de expressar corretamente as restrições de invocação das chamadas de serviços Web de um documento AXML. Isto se deve principalmente à ocorrência de dependências compartilhadas e de seqüências automáticas de invocações. Nesta tese, é apresentado um formalismo de representação baseado em grafos orientados acíclicos (mais conhecidos como DAG, de "*Directed Acyclic Graphs*"), inspirado nos modelos utilizados em sistemas de *workflows* (AALST *et al.*, 2003, EROL *et al.*, 1994, HOLLINGSWORTH, 1995, LIU *et al.*, 2004b). Este formalismo captura as

características mais relevantes da materialização de um documento AXML em um grafo de dependências. São apresentados critérios para simplificar redundâncias e verificar a validade de um grafo de dependências em relação à existência de ciclos infinitos de execução e bloqueios em espera (*i.e.*, "*deadlocks*"). A simplificação de redundâncias é importante para se obter um modelo canônico de representação. A partir deste modelo, são propostas alternativas eficientes para atualizar um grafo de dependências com respostas ativas durante o processo de materialização.

O cerne da estratégia de otimização desta tese consiste em um algoritmo capaz de lidar com documentos AXML arbitrariamente complexos. Este algoritmo usa o grafo de dependências de um documento AXML para quebrar o problema de materialização em partes menores. Iterativamente, ele intercala planejamento e execução, permitindo assim que o sistema produza planos de materialização parciais e entregue os resultados correspondentes antecipadamente. Tais planos são descritos com operadores de uma álgebra de materialização, a qual contempla uma avaliação incremental e descentralizada de chamadas de serviços Web. Esta abordagem dinâmica possui várias vantagens. Em particular, ela reduz a complexidade da seleção de sítios e permite que o otimizador disponha de informações mais atualizadas sobre o sistema, contribuindo para uma maior qualidade das estatísticas e para a validade do plano. Além disso, a localização dos provedores dos serviços Web requisitados pode ser postergarda até que seja realmente necessária. A estratégia proposta também é adaptativa, como definido em HELLERSTEIN *et al.* (2000), já que permite ao otimizador considerar as eventuais mudanças no sistema P2P a cada passo do processo de otimização. São gerados planos parciais (possivelmente independentes entre si) que podem ser distribuídos no sistema P2P, alavancando a colaboração entre sítios. A seleção dos sítios tanto para a execução como para a coordenação das tarefas é automática e considera os custos envolvidos na migração do controle de execução.

A execução eficiente de *workflows* distribuídos com coordenação descentralizada ainda representa um desafio para a pesquisa. No sistema GRAND, VARGAS *et al.* (2004, 2005) propõem uma arquitetura baseada em redes hierárquicas para distribuir as submissões de tarefas de um *workflow*. Contudo, a escolha dos sítios envolvidos nessa distribuição é aleatória. Por outro lado, o sistema GridFlow (CAO *et al.*, 2003) usa uma rede hierárquica para executar *workflows* distribuídos, com

otimização descentralizada. Porém, não são considerados custos de transferência de dados entre tarefas e a otimização é estática. O sistema MENTOR (WODTKE *et al.*, 1996, MUTH *et al.*, 1998) é um dos pioneiros em considerar *workflows* descentralizados, permitindo que partes da execução de um *workflow* sejam distribuídas entre diferentes controladores. Entretanto, o foco é garantir a correção na divisão de *workflows* em partes menores, ignorando a seleção dos melhores controladores para cada parte.

Já no sistema ADEPT-WfMS (BAUER e DADAM, 1997, 2000), é apresentado um algoritmo para delegar tarefas entre servidores baseado em uma análise de probabilidade. Considera-se a possibilidade de cada tarefa ser executada em um determinado sítio, a partir de especificações fornecidas pelo usuário. O ADEPT-WfMS visa reduzir as transferências de dados na execução de um *workflow* distribuído. Entretanto, a geração de planos é centralizada, estática e depende do usuário para determinar a divisão do *workflow* e mapear tarefas em sítios. Como sistemas P2P são arranjos complexos de recursos, é importante que a geração de planos de execução seja automática. Esse ponto é evidenciado por VIEIRA (2005), que propõe mecanismos para a flexibilização da execução de *workflows*, visando o tratamento de exceções.

No contexto de *grids*, ARORA *et al.* (2002) descrevem um algoritmo descentralizado para o escalonamento de tarefas. Porém, os autores consideram apenas tarefas independentes e sem transferências de dados. Definido em um cenário P2P, o sistema *Self-Serv* (BENATALLAH *et al.*, 2005) também propõe uma coordenação descentralizada de tarefas. Todavia, é adotada uma estrutura hierárquica, na qual um coordenador geral delega tarefas aleatoriamente. Vale ressaltar que os sistemas GRAND, MENTOR, ADEPT-WfMS e *Self-Serv* pressupõem que apenas um conjunto fixo de servidores são capazes de controlar a execução de tarefas. Além disso, o processo de otimização é centralizado e não há possibilidade de delegação transitiva (*i.e.*, de um controlador decidir repassar subpartes do problema para outros controladores).

Muitos dos otimizadores para a execução de *workflows* em ambientes distribuídos são baseados na geração estática de planos (ALHUSAINI *et al.*, 1999, BAUER e DADAM, 1997, BLYTHE *et al.*, 2005, BRAUN *et al.*, 2001, CAO *et al.*, 2003, GOUNARIS *et al.*, 2004b, VARGAS *et al.*, 2005, WIECZOREK *et al.*, 2005,

ZENG *et al.*, 2004, ZHANG *et al.*, 2006). Mesmo quando essa geração é dinâmica, as abordagens de otimização costumam considerar tarefas independentes (ARORA *et al.*, 2002, DAGMAN, 2006, FOSTER, 2002, 2003, HUANG *et al.*, 2005) e são baseadas em decisões locais (OINN *et al.*, 2004, THAIN *et al.*, 2005, TRIANA, 2003, TAYLOR *et al.*, 2003) ou oportunistas (MEYER *et al.*, 2006b, SINGH *et al.*, 2006). Os algoritmos de agendamento de tarefas desses otimizadores são aplicados apenas às tarefas prontas para execução, cujo agrupamento é geralmente restrito a tarefas independentes entre si. Ou seja, o otimizador só costuma avaliar transferências de dados entre tarefas relacionadas após cada etapa de execução, visando rodar as próximas tarefas em sítios que já possuam os dados necessários. Porém, na materialização de um documento AXML, os resultados das chamadas de serviços Web são retornados para o sítio que as invocou (*i.e.*, o sítio mestre, caso a invocação das chamadas não seja delegada a outros sítios). Assim, para que haja redução de custos de transferência de dados, a delegação de chamadas de serviços Web com dependências entre si deve ser decidida *a priori*. Portanto, algoritmos de otimização para *workflows* ignoram os benefícios da delegação da invocação de serviços Web, especialmente quanto aos custos de comunicação.

Para avaliar planos de materialização equivalentes, é apresentado nesta tese um modelo de custo que considera as características típicas de um sistema P2P, tais como: máquinas e redes de comunicação heterogêneas; uso de serviços Web equivalentes; execução distribuída; e delegação de chamadas de serviços Web. Além disso, o modelo comporta diferentes heurísticas para agendamento de tarefas. São propostas fórmulas que refletem aspectos específicos de serviços Web, incluindo operações como empacotamento de mensagens SOAP, validação (*i.e.*, "*parsing*") e serialização de dados XML. Em RUBERG *et al.* (2004a, 2004b), verificou-se em uma análise empírica que essas operações podem apresentar um alto custo computacional no acesso a serviços Web. Contudo, esses fatores têm sido negligenciados nos modelos de custo para sistemas P2P e *grids* (ALHUSAINI *et al.*, 1999, BAUER e DADAM, 2000, BLYTHE *et al.*, 2005, YU *et al.*, 2005), mesmo em plataformas orientadas a serviços Web (AZEVEDO, 2003, BENATALLAH *et al.*, 2005, ZENG *et al.*, 2004).

Devido ao caráter descentralizado e não determinístico de um ambiente P2P, a representação da carga de processamento dos sítios é bastante complexa na materialização de documentos AXML. Porém, tal representação é essencial na seleção

dos melhores recursos para executar um plano de materialização. Nesta tese, para um cômputo mais realista dessa carga, é utilizado um modelo flexível de filas de processamento, no qual o agendamento de tarefas não precisa estar totalmente definido. Os modelos da literatura consideram um agendamento de tarefas definido globalmente e *a priori* (BLYTHE *et al.*, 2005, KWOK e AHMAD, 1999, WIECZOREK *et al.*, 2005). Isto resulta em um número enorme de planos alternativos a serem avaliados.

A técnicas de otimização propostas nesta tese apresentam inovações em uma ampla gama de problemas relacionados à execução de *workflows* em sistemas distribuídos e heteregêneos. Em uma abordagem sistemática, são considerados aspectos cruciais, como a descentralização e a volatilidade dos sistemas P2P. Embora a estratégia proposta beneficie a materialização de documentos AXML em geral, ela favorece especialmente documentos cujas chamadas embutidas referenciam serviços Web com foco em dados. Isto é, serviços que envolvam a transferência de volumes substanciais de dados, seja nos parâmetros de entrada ou nos resultados. É interessante destacar que neste cenário o principal objetivo não é achar planos ótimos, mas principalmente evitar as piores soluções. A estratégia proposta favorece também a robustez do processo de materialização de documentos AXML. Planos de materialização podem falhar devido a diversas razões, tais como restrições de segurança e erros de *hardware*. Contudo, uma estratégia dinâmica e incremental contribui para a recuperação eficiente do sistema, pois tarefas menores podem ser mais facilmente monitoradas.

Por fim, as técnicas propostas foram incorporadas no **XCraft**, um otimizador colaborativo baseado em uma arquitetura orientada a serviços, construído na plataforma *ActiveXML*. Também são apresentados métodos para coletar os parâmetros necessários à análise de custo de planos de materialização. De modo uniforme, o XCraft explora serviços Web para colaboração entre sítios, tanto na coleta de informações como na otimização de planos de materialização. A eficiência da estratégia proposta é avaliada por meio de resultados experimentais reais e simulados, obtidos com um protótipo do XCraft. São utilizados arquivos sintéticos do *benchmark* XMark (XMARK, 2003) e documentos AXML com diversas topologias, explorando padrões típicos de *workflows* e sistemas P2P. Em vários casos, o XCraft consegue obter ganhos significativos de desempenho, especialmente quando ocorre a delegação de tarefas.

1.4 ORGANIZAÇÃO DO TEXTO

Esta tese está organizada em oito capítulos adicionais, como se segue.

O Capítulo 2 ilustra, por intermédio de uma aplicação financeira, os conceitos básicos e as principais questões envolvidas na materialização de documentos AXML. Documentos AXML são formalmente definidos neste capítulo, com enfoque na especificação de chamadas de serviços Web. Também são descritos os componentes da plataforma *ActiveXML* e é feita uma análise da literatura sobre a materialização de documentos AXML. Já no Capítulo 3, são abordadas técnicas para a execução eficiente de *workflows* em ambientes distribuídos e heterogêneos, incluindo um estudo sobre o estado da arte dos principais otimizadores e trabalhos correlatos.

Os conceitos fundamentais da estratégia de otimização proposta nesta tese são apresentados no Capítulo 4, o qual introduz um formalismo canônico para representar as restrições de invocação das chamadas serviços Web de um documento AXML. Essas restrições são sintetizadas em grafos de dependências, para os quais são especificadas regras para verificação de validade, eliminação de redundâncias e atualização dinâmica eficiente com respostas ativas. São definidas as possibilidades de execução das chamadas de serviços Web de um documento AXML em um sistema P2P. A partir deste formalismo, o Capítulo 5 descreve a proposta desta tese: uma estratégia dinâmica e descentralizada para a materialização eficiente de documentos AXML. São apresentadas técnicas para geração de planos de materialização para grafos de dependências, incluindo uma álgebra de operadores e um algoritmo dinâmico para a materialização incremental de documentos AXML.

O Capítulo 6 apresenta uma avaliação empírica dos resultados obtidos com um protótipo do XCraft, um otimizador orientado a serviços Web que implementa as técnicas propostas nesta tese. Essa avaliação inclui um estudo dos fatores que afetam a complexidade da busca por soluções eficientes na materialização de documentos AXML e uma análise dos ganhos de desempenho obtidos com a delegação de chamadas de serviços Web. Por fim, o Capítulo 7 destaca as conclusões gerais e contribuições desta tese, bem como algumas considerações finais e perspectivas de pesquisa.

Capítulo 2

FUNDAMENTOS DE DOCUMENTOS AXML

Este capítulo ilustra, por meio de uma aplicação financeira, as questões envolvidas na materialização de documentos AXML. Em seguida, é detalhada a representação formal desses documentos, bem como a plataforma P2P ActiveXML. Por fim, são discutidos os principais trabalhos da literatura sobre técnicas de otimização para a materialização de documentos AXML.

2.1 INTRODUÇÃO

Documentos XML ativos (ou documentos AXML – de "*Active XML*") exploram a idéia de integrar dados e programas em um modelo uniforme (ABITEBOUL *et al.*, 2003a, 2003b, 2004c, ABITEBOUL RUBERG e NGUYEN, 2006b, RUBERG *et al.*, 2004a, PEREIRA, RUBERG e MATTOSO, 2006), que sempre representou um desafio importante na pesquisa em Banco de Dados. O modelo de documentos AXML foi inspirado na materialização de visões em bancos de dados dedutivos (ABITEBOUL *et al.*, 1999). Nesses sistemas, os usuários podem definir regras que disparam automaticamente uma ação quando satisfeitas determinadas condições. Tais regras são armazenadas no banco de dados, podendo ser consultadas e manipuladas declarativamente. Além disso, também é possível definir uma ordem de aplicação dessas regras, em um sistema semelhante à execução de *workflows* (WIDOM, 1996,).

Com a consolidação dos padrões XML e de serviços Web (WEB SERVICES, 2006), os princípios gerais desses mecanismos de regras foram utilizados nos documentos AXML visando superar as barreiras da heterogeneidade de recursos e permitir a manipulação declarativa de execuções de programas (*i.e.*, de serviços Web). Documentos AXML usam a tecnologia de serviços Web para encapsular em uma interface uniforme diversas implementações de programas, oferecendo meios para integrar informações armazenadas em fontes distribuídas, heterogêneas e autônomas. Nesses documentos, elementos especiais representam chamadas a serviços Web, constituindo assim uma mídia particularmente interessante para representar informações voláteis, que precisam ser atualizadas freqüentemente.

Várias aplicações que tiram proveito de documentos AXML já foram apresentadas, como a divulgação de notícias no formato RSS (ABITEBOUL *et al.*, 2003a) e a gerência distribuída de informações médicas (ABITEBOUL *et al.*, 2004a). Recentemente, documentos AXML têm sido utilizados no projeto europeu EDOS (EDOS, 2004), na gerência de distribuição de *software* livre (ABITEBOUL *et al.*, 2005d, ABITEBOUL *et al.*, 2006a). Essas aplicações são implementadas em sistemas P2P.

Como parte dos dados de um documento AXML é virtualmente representada por chamadas de serviços Web, para acessar o conteúdo completo é preciso antes materializá-lo por meio da invocação das chamadas de serviços embutidas no documento. Esta tese trata da otimização de desempenho da materialização de documentos AXML em sistemas P2P. No desenvolvimento desta tese, o uso de documentos AXML foi investigado em dois cenários principais: na integração de dados científicos e em aplicações financeiras. No sistema *Acware* (ABITEBOUL, RUBERG e NGUYEN, 2006b), documentos AXML são usados na implementação de um repositório de conteúdo ativo, para auxiliar biólogos a integrar e transformar continuamente informações sobre controle de risco alimentar. Por intermédio de serviços Web, um repositório de conteúdo ativo reúne dados advindos de fontes distribuídas e heterogêneas. O conteúdo é dito ativo porque os dados são armazenados no formato AXML, cujas chamadas de serviços Web visam alimentar e refinar o repositório. Um diferencial importante do sistema *Acware* consiste na geração declarativa dessas chamadas de serviços Web, a partir de consultas sobre dados armazenados no repositório.

Vale ressaltar que um repositório de conteúdo costuma reunir um considerável número de chamadas de serviços Web, em um ambiente dinâmico cujos participantes podem entrar e sair do sistema aleatoriamente. Assim, o sistema *Acware* evidencia a necessidade do uso de referências abstratas para serviços Web e da geração automática de planos de materialização para documentos AXML.

No contexto de aplicações financeiras, RUBERG *et al.* (2004a, 2004b) descrevem uma implementação baseada em documentos AXML para controle e acompanhamento do Programa Nacional de Fortalecimento da Agricultura Familiar (PRONAF), do Banco Nacional de Desenvolvimento Econômico e Social (BNDES). O PRONAF trata de empréstimos bancários para apoio ao desenvolvimento rural e envolve participantes como o próprio BNDES, o Banco Central do Brasil e vários agentes financeiros. Neste cenário, documentos AXML representam os principais elementos de integração entre os participantes do PRONAF, oferecendo também mecanismos flexíveis para lidar com a volatilidade das informações financeiras.

Uma característica interessante das aplicações financeiras decorre do fato de que desempenho é tão importante quanto outros aspectos não-funcionais tradicionalmente

críticos, como robustez e segurança. Por exemplo, em sistemas para negociação de ações, o fechamento de bons negócios depende de consultas a dados que sejam os mais atuais possíveis, para subsidiar a tomada de decisões. Como aplicações financeiras geralmente ocorrem em cenários distribuídos e altamente dinâmicos, técnicas de otimização são um forte requisito dos sistemas que as apoiam.

Neste capítulo são abordadas as principais questões que afetam o desempenho da materialização de documentos AXML, bem como algumas oportunidades de otimização. Na seção 2.2, o uso de documentos AXML para a integração de informações em um sistema P2P é ilustrado por meio de um exemplo de aplicação financeira para acompanhamento de operações de *Swap* cambial. Este exemplo será utilizado como referência ao longo da tese. Em seguida, na seção 2.3, documentos AXML são formalmente definidos a partir dos conceitos introduzidos em MILO *et al.* (2003) e ABITEBOUL *et al.* (2004c), com ênfase na especificação de chamadas de serviços Web. Por fim, a seção 2.4 descreve a plataforma P2P *ActiveXML* (ACTIVEXML, 2002, ABITEBOUL *et al.*, 2004c, 2005c) e apresenta uma análise da literatura sobre técnicas de otimização para a materialização de documentos AXML.

2.2 CASO DE USO: CONTROLE DE *SWAP* CAMBIAL

A Figura 1 mostra o diagrama de caso de uso de uma aplicação para controle de operações de *Swap* cambial, chamada *CurrencySwap*. Basicamente, uma operação de *Swap* cambial consiste em trocar títulos de dívida em uma determinada moeda por uma outra moeda, ou por uma taxa de juros pré-fixada (BACEN, 2006). Isto permite que instituições financeiras adquiram garantias contra variações adversas de cotação cambial.

Por exemplo, suponha que uma empresa tenha adquirido um empréstimo cujo valor está especificado em dólares. Para evitar surpresas indesejáveis no vencimento deste empréstimo, a empresa pode negociar com um banco de investimentos a troca da dívida em moeda estrangeira por uma dívida com um taxa fixa de juros. Assim, na operação de *Swap*, a dívida original (em Dólares) é convertida para Reais, de acordo com a taxa de câmbio da data de contratação do *Swap*. No vencimento do empréstimo, se a variação da cotação do Dólar for maior que a taxa negociada na operação de *Swap*,

caberá ao banco pagar a diferença para liquidar o empréstimo. Por outro lado, a empresa deverá arcar com a taxa da operação de *Swap* mesmo que a variação da cotação do Dólar seja inferior. Em geral, empresas recorrem a agentes financeiros para contratar operações de *Swap*, as quais são negociadas em uma bolsa de valores.

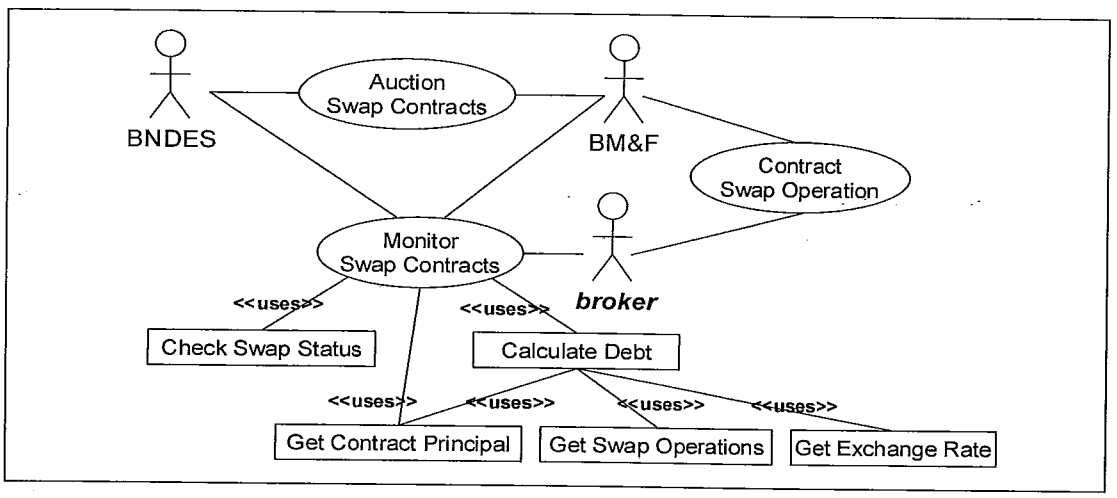


Figura 1. Diagrama de caso de uso da aplicação *CurrencySwap*.

A Figura 1 mostra uma visão geral da aplicação *CurrencySwap*. Os atores são os agentes financeiros (*brokers*), o BNDES e a Bolsa de Mercadorias e Futuros (BM&F). Supomos que operações de *Swap* são negociadas apenas com taxas pré-fixadas e que as empresas interessadas em contratos de *Swap* são sempre representadas por *brokers*. A BM&F gerencia todas as operações de *Swap* solicitadas pelos *brokers*. Essas operações são negociadas com o BNDES, o banco de investimento. Por sua vez, o BNDES deve limitar o valor máximo de cada operação de *Swap* por empresa, visando assim reduzir seu risco financeiro. Para isso, o BNDES atribui um estado a cada empresa, indicando se ainda existe margem disponível para contratação de operações de *Swap*.

2.2.1 DOCUMENTO ATIVO *SWAPWORKSPACE*

Durante as negociações, *brokers* devem monitorar continuamente os contratos de *Swap* de suas respectivas empresas. Contudo, tais informações são voláteis e estão distribuídas entre os participantes do sistema. Portanto, são necessárias técnicas de integração virtual de dados para reunir essas informações adequadamente. Documentos AXML são uma alternativa interessante para representar informações voláteis e distribuídas, pois adotam uma abordagem híbrida de integração de dados. Nessa

abordagem, a integração virtual ocorre por meio das chamadas de serviços Web. Porém, essas chamadas geram resultados que podem ser armazenados (acumulativamente) no documento. Sempre que necessário, o conteúdo do documento AXML pode ser atualizado pela invocação de suas chamadas de serviços Web. Além disso, documentos AXML consistem essencialmente de dados XML, para os quais já existe uma vasta gama de ferramentas de manipulação. Isso favorece a portabilidade desses documentos entre participantes de um sistema heterogêneo.

A Figura 2 exibe o documento *SwapWorkspace* em uma notação AXML simplificada (a linguagem completa é descrita em ACTIVEXML TEAM, 2005), que é usado pelos *brokers* para acompanhar as informações de um determinado contrato de *Swap*. O conteúdo desse documento inclui: o número do contrato de *Swap*; o nome da empresa contratante e seu estado junto ao BNDES; o montante principal da dívida em moeda estrangeira; o valor correspondente convertido conforme as operações de *Swap*; a data corrente; e um extrato dos termos do contrato negociado. Essas informações são obtidas a partir de serviços Web publicados pelos participantes do sistema. As chamadas de serviços Web embutidas no documento *SwapWorkspace* são representadas por elementos "sc". Ao longo do texto, chamadas de serviços Web são indicadas por scX, onde X é o valor do atributo "id" do respectivo elemento "sc" (*e.g.*, sc1).

No documento *SwapWorkspace* da Figura 2, um *broker* precisa determinar apenas o número de contrato de *Swap* a ser monitorado e o nome da empresa correspondente. Com essas informações, cabe ao sistema (sob demanda ou periodicamente) atualizar o conteúdo do documento AXML, invocando adequadamente as chamadas de serviços Web embutidas no documento. Note que o resultado de algumas chamadas de serviços Web podem ser usados para alimentar uma outra chamada, como acontece com sc9 e sc10. Neste caso, a chamada sc9 depende de sc10.

```
<current_contract>
.... <number> 12345 </number>
.... <company> <name>XTechno Acme Ltd</name>
.... .... <can_swap>
.... .... .... <sc id="1" service="CheckSwapStatus">
.... .... .... .... <param name="swaps">
.... .... .... .... .....<sc id="2" service="GetCurrentSwaps">
.... .... .... .... .....<xpath>/current_contract/company/name</xpath>
.... .... .... .... .....</sc>
.... .... .... .... </param>
.... .... .... .... <param name="current_limit">
.... .... .... .... .....<sc id="3" service="GetSwapLimit">
.... .... .... .... .....<param name="company">
.... .... .... .... ..... ....<xpath>/current_contract/company/name</xpath>
.... .... .... .... .....</param>
.... .... .... .... .....<param name="date">
.... .... .... .... ..... ....<xpath>/current_contract/today</xpath>
.... .... .... .... .....</param>
.... .... .... .... .....</sc>
.... .... .... .... </param> </sc>
.... .... </can_swap> </company>
.... <principal>
.... .... <sc id="4" service="GetContractPrincipal">
.... .... .... <xpath>/current_contract/number</xpath>
.... .... </sc>
.... </principal>
.... <swap_debt>
.... .... <sc id="5" service="CalculateDebt" followed_by="1">
.... .... .... <param name="principal">
.... .... .... .... <xpath>/current_contract/principal/amount</xpath>
.... .... .... </param>
.... .... .... <param name="swaps">
.... .... .... .... <sc id="6" service="GetSwapOperations">
.... .... .... .... .....<xpath>/current_contract/number</xpath>
.... .... .... .... </sc>
.... .... .... </param>
.... .... .... <param name="rate">
.... .... .... .... <sc id="7" service="GetExchangeRate">
.... .... .... .... .....<param name="foreign_currency">
.... .... .... .... .....<xpath>/current_contract/principal/currency</xpath>
.... .... .... .... .....</param>
.... .... .... .... .....<param name="date">
.... .... .... .... .....<xpath>/current_contract/today</xpath>
.... .... .... .... .....</param>
.... .... .... .... </sc>
.... .... .... </param> </sc>
.... </swap_debt>
.... <today> <sc id="8" service="GetLocalDate"/> </today>
.... <contract_excerpt>
.... .... <sc id="9" service="ExtractExcerpt">
.... .... .... <param name="text"> <sc id="10" service="GetContractPDF"/> </param>
.... .... .... <param name="input format">PDF</param>
.... .... .... <param name="output_format">XML</param>
.... .... </sc>
.... </contract_excerpt>
</current_contract>
```

O conteúdo dos elementos "sc" deve ser recuperado dinamicamente

Parâmetros de entrada podem ser definidos por expressões XPath

Este atributo indica uma chamada colateral a sc1

Figura 2. Documento AXML *SwapWorkspace.*

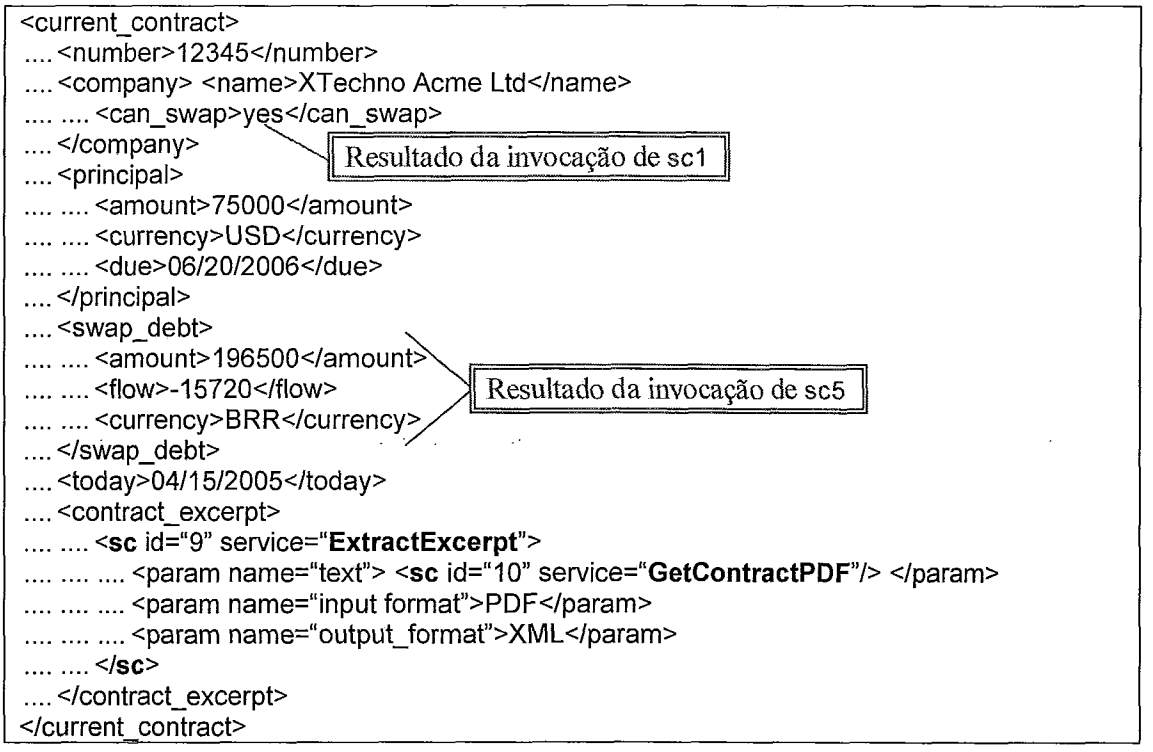


Figura 3. Versão parcialmente materializada do documento ativo *SwapWorkspace* após a invocação de sc5.

Uma versão parcialmente materializada do documento *SwapWorkspace* é mostrada na Figura 3, cujo conteúdo é obtido após a invocação da chamada de serviço sc5. Note que com sc5 também são invocadas suas dependências (sc6 e sc7) e conseqüências (sc1, conforme indicado pelo atributo "followed_by"), bem como todas as chamadas de serviços envolvidas transitivamente (sc2, sc3, sc4 e sc8). O elemento "flow" indica a diferença financeira entre a variação cambial e a taxa pré-fixada da operação de *Swap*. Ou seja, quando "flow" é positivo, há prejuízo para o banco de investimento. Note que as chamadas de serviços sc9 e sc10 não são invocadas pois não são dependências transitivas nem chamadas colaterais transitivas de sc5.

Na Figura 3 constam apenas os elementos XML regulares que formam o conteúdo do documento *SwapWorkspace*. Porém, é possível manter no documento AXML as chamadas de serviços Web mesmo após o processo de materialização. De fato, os resultados das chamadas de serviços Web são inseridos no documento AXML conforme especificado nos respectivos elementos "sc". Isto é, seja: substituindo a chamada de serviço Web; ou inserindo o resultado como nodo(s) irmão(s) da chamada de serviço. Em geral, pode-se também descobrir antecipadamente os tipos de elementos

retornados no resultado da execução de um serviço Web por intermédio de sua descrição WSDL.

Vale ressaltar que, embora o documento *SwapWorkspace* represente informações de apenas um contrato de *Swap*, cada *broker* geralmente monitora vários contratos de diversas empresas.

2.2.2 ALTERNATIVAS DE MATERIALIZAÇÃO AXML

Em um cenário P2P, cada participante da aplicação *CurrencySwap* é representado por um sítio capaz de se comunicar diretamente com os demais sítios do sistema. Um sítio também pode prover serviços Web, cujas descrições WSDL podem ser recuperadas do provedor ou de repositórios na rede (*e.g.*, um servidor UDDI). A Figura 4 mostra um exemplo de cenário para a aplicação *CurrencySwap*. Linhas pontilhadas indicam sítios em uma mesma rede local. Por exemplo, os sítios P_4 e P_5 são vizinhos na rede da BM&F. Para maior clareza de apresentação, são omitidas as ligações entre sítios de uma rede local. Neste exemplo, supõe-se que a transferência de dados em uma rede local é 50 vezes mais rápida que em uma conexão pela Internet.

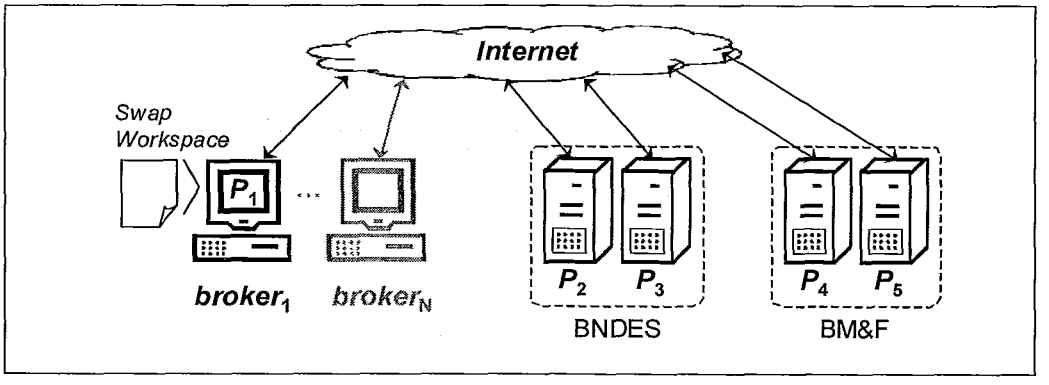


Figura 4. Cenário P2P da aplicação CurrencySwap.

Na Figura 4, o sítio P_1 é dito sítio mestre do documento *SwapWorkspace*. Isto significa que o resultado da materialização deste documento deve ser retornado para P_1. Entretanto, o sítio P_1 pode delegar a materialização de partes do documento AXML a outros sítios, recebendo apenas os resultados que constituem o conteúdo final.

É interessante observar que o documento *SwapWorkspace* da Figura 2 contém referências abstratas para indicar os serviços Web requisitados (*e.g.*, *GetSwapStatus* e

GetExchangeRate). Essas referências são utilizadas em vez de endereços completos, os quais contêm URL, nome de operação e demais parâmetros necessários para executar um serviço Web. No Capítulo 4, são apresentadas várias vantagens do uso de referências abstratas para serviços Web. Contudo, para invocar uma chamada de serviço Web, referências abstratas devem ser traduzidas em endereços completos (ou em um conjunto de endereços completos equivalentes). Para isso, devem-se identificar quais sítios provêem cada serviço Web requisitado pelas referências abstratas do documento AXML.

Para a aplicação *CurrencySwap*, a Tabela 1 mostra a distribuição dos serviços Web (*i.e.*, das referências abstratas de serviços) nos sítios da Figura 4. Vale ressaltar que em sistemas P2P cada serviço Web costuma ser provido por vários sítios.

Tabela 1. Distribuição de serviços Web da aplicação *CurrencySwap*.

sc	Serviço Web	Sítios Provedores
1	*CheckSwapStatus*	P_2, P_3, P_4
2	*GetCurrentSwaps*	P_4, P_5
3	*GetSwapLimit*	P_2, P_3, P_4
4	*GetContractPrincipal*	P_4, P_5
5	*CalculateDebt*	P_2, P_3, P_4, P_5
6	*GetSwapOperations*	P_4, P_5
7	*GetExchangeRate*	P_2, P_3, P_4, P_5
8	*GetLocalDate*	P_1, P_2, P_3, P_4, P_5
9	*ExtractExcerpt*	P_1, P_4, P_5
10	*GetContractPDF*	P_4, P_5

Os elementos "sc" do documento *SwapWorkspace* apontam para serviços Web que são providos por vários sítios, conforme a Tabela 1. Portanto, para materializar o documento, é necessário determinar exatamente qual sítio deverá tratar cada chamada de serviço Web. Ou seja, para cada chamada embutida no documento AXML, o sistema deve identificar os provedores e então escolher o melhor sítio para executar o serviço Web correspondente. De acordo com os executores escolhidos e as possibilidades de delegação de chamadas de serviços, várias alternativas de avaliação podem ser adotadas para materializar um documento AXML. Cada alternativa pode ser representada por um plano de materialização.

A Figura 5 ilustra uma alternativa centralizada, cujo extrato é representado a seguir (de acordo com a distribuição de serviços da Tabela 1), para materializar o conteúdo representado pela chamada de serviço sc9 da Figura 2. Observe que sc9 recebe como entrada o resultado de sc10 no documento *SwapWorkspace*. Portanto, para invocar sc9, deve-se primeiro materializar o resultado de sc10. Na avaliação centralizada, o sítio mestre (*i.e.*, P_1) invoca tanto sc10 como sc9.

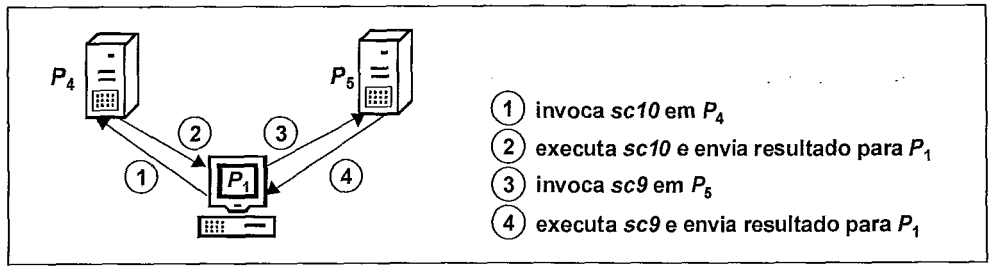
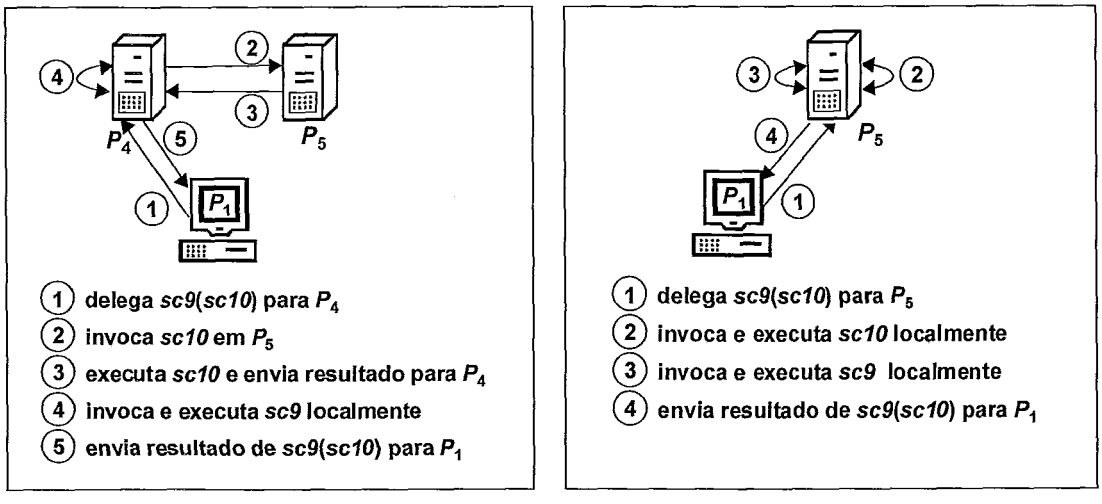


Figura 5. Materialização AXML por meio de avaliação centralizada.

Já na Figura 6, são exibidas algumas alternativas descentralizadas, também de acordo com a Tabela 1, para a materialização da chamada de serviço sc9. Ou seja, o sítio P_1 delega a invocação das chamadas de serviços sc9 e sc10 seja para P_4 (Figura 6.a) ou P_5 (Figura 6.b). Nestes casos, pressupõe-se que os sítios P_4 e P_5 são capazes de colaborar com P_1, invocando as chamadas de serviços Web e repassando os resultados necessários para P_1.



(a) (b)

Figura 6. Algumas alternativas descentralizadas para materialização AXML.

2.2.3 Oportunidades de Otimização

Para materializar um documento AXML inteiro, a escolha do executor e da delegação de uma chamada de serviço Web é geralmente influenciada pela invocação de outras chamadas do documento. Essa influência ocorre especialmente quando os parâmetros de entrada das chamadas de serviços Web do documento contêm outros elementos "sc". Por exemplo, a escolha dos sítios envolvidos na materialização de sc10 tem impacto direto no desempenho da materialização de sc9, principalmente no custo de transferência de dados. Note que apenas o resultado de sc9 é realmente necessário para construir o conteúdo do documento *SwapWorkspace*. Logo, considerando que P_4 e P_5 estão em uma mesma rede local, em uma análise intuitiva, a alternativa da Figura 6 permitiria obter ganho de desempenho em relação à Figura 5, por meio da delegação de chamadas de serviços Web. Já na Figura 6.b, as execuções de serviços Web são concentradas no sítio P_5, evitando transferências de dados. Porém, dependendo do restante do documento AXML a ser materializado, nem sempre tal concentração de processamento é interessante, pois não explora execuções paralelas e pode implicar em sobrecarga do sítio. Observe que em alternativas descentralizadas de avaliação, o sítio mestre fica disponível para tratar outras chamadas de serviços Web enquanto aguarda o resultado da delegação.

Na materialização de documentos AXML, a delegação de chamadas de serviços é particularmente interessante quando há dependências de dados chamadas de serviços e os sítios executores dos serviços Web podem se comunicar entre si por um canal mais rápido do que com o sítio mestre. Assim, podem-se reduzir os custos de comunicação. Todavia, para colaborar com outros sítios por meio de delegação da materialização, o sítio mestre deve enviar para os sítios selecionados as chamadas de serviços Web a serem invocadas remotamente, juntamente com os respectivos parâmetros de entrada. Em alguns casos, esses parâmetros podem ser volumosos em relação aos resultados das chamadas de serviços Web, aumentando os custos da delegação. Portanto, é necessário analisar cuidadosamente as possíveis alternativas de materialização de um documento AXML.

O desempenho da materialização de um documento AXML pode variar muito conforme a estratégia de avaliação adotada. No documento *SwapWorkspace*, se a transferência dos resultados de sc10 pela Internet custa 50s (*e.g.*, de P_5 para P_1, como

na Figura 6), então ela custaria apenas 1s em uma rede local, já que este exemplo pressupõe uma relação de 50:1 entre esses enlaces. Ou seja, a delegação consiste em uma alternativa interessante neste exemplo. Para grandes volumes de dados e várias chamadas de serviços Web, tal diferença pode ser ainda mais significativa. Por outro lado, uma estratégia de materialização aleatória ou ingênua pode gerar tempos de execução inaceitáveis.

Outra vantagem da delegação é a possibilidade de explorar execuções paralelas por meio da descentralização do processo de materialização. Para tanto, deve-se determinar uma ordem de invocação eficiente para as chamadas de serviços do documento AXML, considerando as possíveis delegações. Aliás, como a delegação pode descentralizar o processo de materialização, tal ordem de invocação é parcial e visa essencialmente explorar o paralelismo entre chamadas de serviços independentes entre si e balancear a carga de processamento dos sítios. Além da delegação de chamadas de serviços, ganhos de desempenho podem ser obtidos pela escolha dos sítios mais eficientes entre os provedores de cada serviço Web. O desempenho de um sítio na execução de uma chamada de serviço Web é determinado por vários fatores, como a capacidade de processamento do sítio, a velocidade de comunicação com o sítio mestre e a carga de trabalho alocada ao sítio.

Otimizar a materialização de documentos AXML é um problema difícil, pois o número de alternativas de avaliações pode crescer dramaticamente até mesmo para cenários muito simples. Por exemplo, no documento *CurrencySwap* (com apenas 10 chamadas de serviços e 5 sítios envolvidos) existem pelo menos $1{,}898 \times 10^{16}$ planos de materialização alternativos. Considerando uma medida razoável para os computadores pessoais modernos, cada plano pode ser gerado e analisado em 0,5ms. Com isso, uma busca exaustiva duraria cerca de 305 mil anos (*sic*!) para determinar o melhor plano de materialização.

Algumas heurísticas podem reduzir significativamente o número de planos de materialização de um documento AXML. No documento *SwapWorkspace*, pode-se aplicar a clássica heurística de "Dividir para Conquistar" (D&C), conforme explorado em RUBERG *et al.* (2004a), que quebra o problema de materialização em tarefas independentes. Isso reduziria o espaço de busca para aproximadamente $1{,}265 \times 10^{14}$ planos alternativos. Porém, isto significa mais de dois mil anos somente para escolher o

plano de materialização, para então materializar efetivamente o documento. Apesar da grande melhora em relação à busca exaustiva simples, o tempo gasto na otimização permanece crítico. Vale ressaltar também que este exemplo possui apenas cinco sítios distintos. Em uma configuração com dez sítios, mesmo aplicando a heurística D&C, o espaço de busca se torna 4096 vezes maior. Na seção 5.2, são apresentadas fórmulas para estimar o número de planos de materialização de documentos AXML.

O tempo gasto em otimização pode facilmente se tornar inaceitável na materialização de documentos AXML. De fato, em cenários típicos de materialização AXML, métodos exaustivos e ótimos em geral são inviáveis. Principalmente, porque cenários P2P são bastante dinâmicos e a configuração dos sítios participantes do sistema pode mudar durante a otimização. Conseqüentemente, torna-se imperativo reduzir o espaço de busca para um tamanho que seja tratável e preferencialmente independente do tamanho do problema (*i.e.*, do documento AXML). Para comparar planos de materialização, são necessárias métricas de desempenho que considerem as principais variáveis do problema, como a execução de serviços Web, a delegação de tarefas e os diferentes padrões de dependências entre as chamadas de serviços de um documento AXML. Estratégias puramente gulosas são míopes em relação a esses fatores e costumam gerar soluções com desempenho insatisfatório.

2.3 MODELO DE DOCUMENTOS AXML

O uso de XML para representar chamadas de serviços Web pode ser encontrado em várias linguagens, como a *Apache Jelly* (JELLY, 2006) e a *Microsoft* XAML (GRIFFITHS, 2004). Porém, essa idéia foi amplamente expandida para o contexto dos sistemas P2P com os documentos AXML, criando possibilidades inovadoras para o desenvolvimento de sistemas de integração de informações. Documentos AXML provêem visões uniformes sobre informações distribuídas, as quais podem ser materializadas parcial e progressivamente. Por serem especificadas por um vocabulário XML, em geral essas visões podem ser facilmente interpretadas por sítios de um sistema P2P, constituindo assim um poderoso mecanismo de colaboração. Nesta seção, são descritos os principais recursos da linguagem AXML a partir dos conceitos apresentados em ABITEBOUL *et al.* (2004c, 2004d, 2005c), BENJELLOUN (2004) e MILO *et al.* (2003).

Basicamente, a linguagem para especificação de documentos AXML é baseada em um vocabulário XML estendido, o qual contém marcadores especiais para codificar chamadas de serviços Web. Assim, um documento AXML é representado tipicamente por uma árvore rotulada com dois tipos de nodos:

(i) nodos de dados, ou nodos XML regulares, os quais são rotulados seja com nomes de elementos ou com valores de dados, no caso de nodos-folhas; e

(ii) nodos intencionais ou chamadas de serviços Web, os quais codificam toda a informação necessária para identificar um serviço Web, passar os respectivos parâmetos de entrada e inserir seus resultados no documento AXML.

Para uma definição formal de documentos AXML, sejam os seguintes conjuntos: **N** de nodos; **E** de nomes de elementos; **V** de valores de dados; e **S** de nomes de serviços. Esses conjuntos representam domínios disjuntos e possivelmente infinitos.

DEFINIÇÃO 1: Um documento AXML D consiste na expressão $\langle \tau, \lambda \rangle$, onde $\tau = \langle N, A, < \rangle$ é uma árvore ordenada que contém um conjunto finito $N \subset \mathbf{N}$ de nodos, um conjunto de arestas $A \subset N \times N$ e uma função $<$ que determina uma ordem total nos filhos de cada nodo de N. Além disso, a função λ: $N \rightarrow \mathbf{E} \cup \mathbf{V} \cup \mathbf{S}$ rotula os nodos de N, tal que apenas os nodos-folhas são mapeados para **V**. Nodos rotulados em **S** são ditos chamadas de serviços de D.

Quanto à sua especificação, um documento AXML deve obedecer as mesmas regras de formação de dados XML (XML, 2006). Ou seja, um documento AXML é bem formado somente se sua representação XML for bem formada. O elemento raiz de um documento AXML D é denotado por $root_D$.

Embora não seja abordado na Definição 1, considera-se que certos nodos de um documento AXML podem possuir atributos. Em particular, atributos podem aparecer somente em nodos cujo rótulo pertença a $\mathbf{E} \cup \mathbf{S}$. Aliás, na linguagem AXML, atributos são utilizados para codificar informações importantes sobre chamadas de serviços Web e os resultados de suas invocações (BENJELLOUN, 2004). Vale ressaltar que essa simplificação na Definição 1 visa apenas focalizar a representação de um documento

AXML em aspectos essenciais. Ou seja, prevalece a diferença entre dados XML regulares (*i.e.*, elementos XML e valores) e dados intencionais (que ainda precisam ser materializados). Portanto, em um documento AXML, atributos são considerados apenas como simples propriedades de nodos.

2.3.1 Conteúdo de um Documento AXML

O conteúdo de um documento AXML pode ser analisado sob diferentes perspectivas. Considerando-se apenas a especificação XML, o conteúdo inativo corresponde a todos os nodos do documento, no qual chamadas de serviços são tratadas como elementos regulares. Já o conteúdo ativo ou intencional é representado pelas chamadas de serviços do documento AXML, com referência aos dados remotos que essas chamadas podem recuperar. Por outro lado, o conteúdo materializado reúne os dados XML regulares originalmente presentes no documento e os resultados da invocação de chamadas de serviços do documento. O conteúdo materializado é completo (ou final) se não houver chamadas de serviços a serem invocadas no documento, senão ele é parcial.

Vale ressaltar que pode ser impossível se obter o conteúdo materializado completo de um documento AXML, se uma chamada de serviço retornar recursivamente outra chamada para o mesmo serviço. Neste caso, por simplicidade, diz-se completo o conteúdo materializado que é obtido após um número arbitrário de invocações da chamada recursiva. Ao longo do texto, salvo se mencionado diferentemente, o conteúdo de um documento AXML indica seu conteúdo materializado completo.

Para construir o conteúdo materializado de um documento AXML, considera-se que resultados de invocações substituem as respectivas chamadas de serviços no documento. Contudo, na prática, nodos intencionais não são eliminados do documento AXML, mas mantidos para futuras invocações (ABITEBOUL *et al.*, 2004d). Porém, pressupõe que a análise da materialização de um documento AXML enfoca o conjunto de chamadas de serviços a serem invocadas em um determinado instante do tempo. Portanto, sem perda de generalidade, são ignorados os nodos intencionais que permanecem no documento após a invocação, contanto que esses nodos não precisem ser invocados novamente durante a materialização.

A materialização de um documento AXML consiste essencialmente em recuperar e construir os dados representados pelo conteúdo ativo. Ou seja, para acessar o conteúdo de um documento AXML, deve-se antes invocar adequadamente as suas chamadas de serviços, coletar os respectivos resultados e compô-los no documento. Esse processo de materialização ocorre em um sistema P2P cujos sítios podem executar e/ou invocar serviços Web.

Em geral, para manter atualizado o conteúdo de um documento AXML, são necessárias várias materializações ao longo do tempo, periódicas ou sob demanda. Assim, pode ser interessante registrar o histórico dessas materializações no próprio documento, armazenando todos os resultados das invocações de nodos intencionais. Para determinar a versão corrente do conteúdo do documento, os elementos resultantes de cada invocação possuem um atributo especial chamado "timestamp". Com isso, para passar os parâmetros de entrada de uma nova invocação, os usuários podem escolher entre considerar todos os resultados presentes no documento ou apenas o último. Por convenção, considera-se que apenas os últimos resultados das chamadas de serviços são utilizados para compor o conteúdo corrente de um documento AXML.

2.3.2 CHAMADAS DE SERVIÇOS WEB

Em um documento AXML, um subconjunto importante de nodos é formado pelas chamadas de serviços Web. Dado um documento D, esse subconjunto é representado por SC_D, tal que:

$$SC_D = \{v \in N \mid \lambda(v) \rightarrow \mathbf{S}\} \ . \qquad (1)$$

Observe que um nome de serviço em **S** corresponde a um endereço de serviço Web, que pode ser: abstrato, quando for baseado em termos de uma taxonomia de serviços; ou concreto, quando estiver acompanhado de todas as informações exigidas pelo protocolo SOAP (*i.e.*, URL, nome de operação, porta, etc.) para acionar um serviço Web. Essas informações podem ser descritas como atributos de uma chamada de serviço. Por exemplo, para usar um endereço concreto, a chamada de serviço sc10 da Figura 2 poderia ser reescrita da seguinte forma:

```
<sc id="10" serviceURL="http://www.p5/services/CurrencySwap"
        serviceNameSpace="CurrencySwap"
        methodName="GetContractPDF"/>
```

onde serviceURL indica a URL do serviço requisitado (neste caso, considerando o sítio P_5), serviceNameSpace informa o espaço de nomes das mensagens SOAP para este serviço e methodName é o nome da operação a ser invocada. Maiores detalhes sobre a sintaxe da linguagem AXML podem ser encontrados em ACTIVEXML TEAM (2005).

Pressupõe-se que todos os nodos intencionais de um documento AXML são identificados unicamente pelo atributo "id". A geração desses identificadores é automática e pode ser feita por diversos métodos de numeração, tais como o ORDPATH e o *Dewey Decimal Coding* (HÄRDER *et al.*, 2007). Além disso, os filhos de um nodo intencional representam parâmetros de entrada a serem passados para o serviço Web no momento da invocação. Esses parâmetros são representados por elementos "param" aninhados nas chamadas de serviços, conforme é mostrado no documento *SwapWorkspace* da Figura 2. Por simplicidade, note que o elemento "param" é omitido na Figura 2 se a chamada de serviço possuir apenas um parâmetro (*e.g.*, a chamada de serviço sc2).

Quando uma chamada de serviço é invocada, as respectivas sub-árvores são usadas para compor o corpo da mensagem SOAP a ser enviada para o serviço Web. Uma facilidade interessante da linguagem AXML decorre da possibilidade de aninhar chamadas de serviços, colocando nodos intencionais nas sub-árvores dos elementos "param". Tal aninhamento pode ser observado em vários pontos do documento *SwapWorkspace*. Por exemplo, as chamadas de serviços sc2 e sc3 estão aninhadas em sc1. Em geral, pressupõe-se que os parâmetros de entrada de uma chamada de serviço devem ser materializados antes montar a mensagem SOAP. Portanto, o aninhamento de nodos intencionais estabelece uma relação de dependência que afeta a ordem de invocação das chamadas de serviços na materialização de um documento AXML. Ou seja, no documento *SwapWorkspace*, os resultados das chamadas de serviços sc2 e sc3 devem ser recuperados antes de invocar sc1. Com isso, determinam-se também fluxos de dados entre invocações de nodos intencionais.

No aninhamento de chamadas de serviços, alguns resultados de invocações são necessários apenas para alimentar outras chamadas de serviços. Isto é, são resultados intermediários que não compõem o conteúdo final do documento AXML. Por outro lado, as chamadas de serviços que constituem esse conteúdo são os principais alvos do processo de materialização do documento. Devido à localização dos elementos intencionais no documento AXML, essas chamadas são consideradas de primeiro nível, como definido a seguir.

DEFINIÇÃO 2: Dado um documento AXML D, uma chamada de serviço $u \in SC_D$ é dita de primeiro nível se e somente se $\lambda(v) \rightarrow \mathbf{E}$ para todo nodo v ascendente de u em D. O conjunto ξ reúne todas as chamadas de serviços de primeiro nível de D, tal que $\xi \subseteq SC_D$.

No documento *SwapWorkspace*, as chamadas de primeiro nível são ξ={sc1, sc4, sc5, sc8, sc9}. Essas chamadas podem ser usadas para estabelecer prioridades na materialização de um documento AXML, para otimizar a obtenção de resultados parciais.

Vale ressaltar que tanto os parâmetros de entrada como os resultados das chamadas de serviços podem conter dados AXML (*i.e.*, podem conter nodos intencionais). Quando isso ocorre, o resultado da chamada de serviço é dito uma resposta ativa. Essa possibilidade é interessante, pois favorece a colaboração entre sítios e a capacidade de adaptação de serviços Web. Por exemplo, seja um serviço Web que fornece a temperatura atual de algumas cidades. Caso seja solicitada a temperatura de uma cidade desconhecida do serviço, em vez de um erro, este serviço poderia retornar como resultado uma chamada para outro serviço Web, cuja abrangência geográfica fosse maior (ou fosse relacionada à cidade desejada). Assim, quem invocou a chamada de serviço original poderia prosseguir a busca pela informação solicitada. Portanto, devido a repostas ativas, o subconjunto SC_D é dinâmico. Ou seja, na materialização de um documento AXML, a especificação do problema pode mudar em tempo de execução.

2.3.3 PARÂMETROS DE ENTRADA DEFINIDOS POR CONSULTAS

Além do aninhamento de elementos, é possível também definir parâmetros de entrada de uma chamada de serviço por intermédio de consultas sobre os elementos do documento AXML. Em particular, os parâmetros de entrada de uma chamada de serviço podem ser especificados por expressões de caminho da linguagem XPath (XPATH, 1999). Por exemplo, isso ocorre na chamada de serviço sc2 na Figura 2. As expressões de caminho são indicadas por elementos "xpath" aninhados em elementos "param". Nesses casos, o parâmetro de entrada é representado por uma consulta que deve ser avaliada quando a chamada de serviço é invocada. O resultado dessa consulta é passado na requisição do serviço Web, como sub-árvore(s) do respectivo elemento "param". Esse recurso permite também determinar fluxos de dados entre chamadas de serviços, conforme é ilustrado nos parâmetros "company" e "date" da chamada de serviço sc3, na Figura 2. Note que as expressões XPath podem ser arbitrariamente complexas, desde que retornem um conjunto de elementos como resultado (mais especificamente, uma floresta).

Entre outras vantagens, parâmetros de entrada definidos por expressões de caminho permitem o compartilhamento de resultados de chamadas de serviços, evitando assim a replicação de nodos dentro do documento AXML. Por exemplo, no documento *SwapWorkspace*, o resultado de sc4 é usado para alimentar as chamadas de serviços sc5 e sc7, sem que sc4 precise ser replicada nos parâmetros de entrada dessas chamadas. Além disso, a chamada de serviço sc4 é invocada apenas uma vez.

Para identificar as chamadas de serviços requisitadas por um determinado parâmetro de entrada, deve-se analisar os elementos relacionados ao caminho descrito na expressão XPath, como apresentado em ABITEBOUL *et al.* (2004b). Essa análise pode se tornar bastante sofisticada, envolvendo a recuperação dos tipos de dados esperados nos resultados de serviços Web, os quais são descritos nos arquivos WSDL (WSDL, 2006) correspondentes.

2.3.4 CHAMADAS COLATERAIS

Na especificação de documentos AXML, também é possível encadear a invocação de chamadas de serviços com um mecanismo semelhante ao acionamento de

gatilhos. O usuário pode determinar que uma chamada de serviço seja disparada automaticamente sempre após a invocação de outra chamada. Neste caso, a chamada disparada é dita colateral, sendo indicada no documento AXML pelo atributo "followed_by". Por simplicidade, considera-se que apenas uma chamada de serviço pode ser referenciada no atributo "followed_by".

Ao contrário de dependências compartilhadas, que envolvem apenas uma invocação, cada referência de chamada colateral implica em uma nova invocação. Na Figura 2, a invocação de sc5 deve disparar automaticamente a chamada de serviço sc1. Ou seja, no documento *SwapWorkspace*, o nodo sc1 é uma chamada colateral de sc5. Isso indica que, ao materializar esse documento, a chamada de serviço sc1 será invocada duas vezes, sendo uma vez devido à própria materialização de sc1 e a outra decorrente da chamada colateral de sc5.

É interessante observar que, embora uma chamada colateral seja invocada necessariamente após a chamada de serviço que a disparou, podem acontecer outras invocações entre essas duas chamadas. Em particular, isso ocorre quando a chamada colateral possui dependências.

2.4 PLATAFORMA *ACTIVEXML*

Uma premissa fundamental do uso de documentos AXML é a existência de um sistema P2P que apóie o armazenamento e a materialização desses documentos. Para tanto, esse sistema requer dois tipos principais de sítios: (*i*) provedores de serviços Web; e (*ii*) gerentes de documentos AXML, capazes armazenar documentos e invocar chamadas de serviços Web. Observe que, ao contrário dos sítios do tipo (*ii*), sítios provedores dispensam qualquer conhecimento específico sobre dados AXML. Essa característica é bem interessante, pois a tecnologia de serviços Web está sendo cada vez mais utilizada, tornando possível ampliar bastante o escopo de um sistema baseado em documentos AXML.

A plataforma *ActiveXML* (ACTIVEXML, 2002) foi desenvolvida no contexto do projeto GEMO (GEMO, 2002), do *Institut de Recherche en Informatique et en Automatique* (INRIA), na França, sob a coordenação do Prof. Serge Abiteboul. Esta

plataforma visa prover mecanismos para incorporar os principais conceitos da linguagem AXML na gerência de documentos. Um aspecto diferencial da plataforma *ActiveXML* é o uso de uma abordagem declarativa para a publicação de serviços Web. Nessa abordagem, em cada sítio, o usuário pode facilmente publicar consultas sobre o repositório local de documentos AXML, tal que cada consulta é encapsulada como um serviço Web.

2.4.1 ARQUITETURA DE SISTEMA

Uma das principais vantagens da plataforma *ActiveXML* consiste em um mecanismo de publicação de serviços declarativos, os quais são consultas parametrizadas sobre documentos AXML. Essas consultas são encapsuladas com interfaces de serviços Web. Serviços declarativos podem ser especificados em uma extensão da linguagem OQL, chamada X-OQL (AGUILERA, 2002), ou em XSL-T, tornando fácil a publicação de serviços Web.

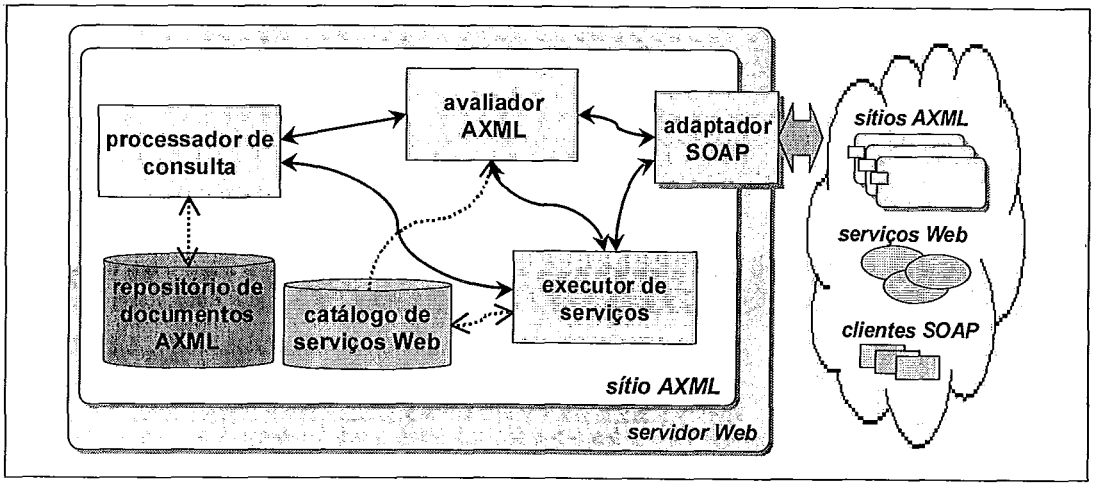


Figura 7. Componentes de um sítio *ActiveXML*

Baseada em uma arquitetura P2P, a unidade fundamental da plataforma *ActiveXML* é um sítio capaz de realizar um conjunto de tarefas básicas para a gerência de documentos AXML. Cada sítio *ActiveXML* pode atuar seja como: um servidor, provendo serviços Web para outros sítios *ActiveXML* ou clientes SOAP; ou como um cliente, quando invoca as chamadas de serviços embutidas em documentos locais, acessando serviços Web remotos. Uma visão simplificada da arquitetura de um sítio *ActiveXML* é exibida na Figura 7. Os principais componentes são:

- Um repositório local de documentos AXML;
- Um processador de consulta, que provê acesso ao repositório local de documentos AXML;
- Um catálogo de serviços Web, contendo as descrições dos serviços providos pelo sítio;
- Um executor de serviços, que executa as requisições direcionadas a serviços providos pelo sítio;
- Um avaliador AXML, para tratar da materialização de nodos intencionais; e
- Um adaptador SOAP, que permite ao sítio tanto se comunicar com outros sítios *ActiveXML*, como invocar chamadas de serviços Web disponíveis na rede.

As mensagens decorrentes de invocações de serviços no sítio *ActiveXML* são devidamente despachadas na rede pelo adaptador SOAP. Além disso, ele recebe as solicitações de execução dos serviços Web que são providos pelo sítio. Essas solicitações são passadas para o executor de serviços declarativos, cujas descrições são armazenadas no catálogo de serviços. É interessante mencionar que, além de serviços declarativos, cada sítio *ActiveXML* oferece um serviço básico de execução de consultas *ad-hoc* no repositório local.

Para tratar requisições de serviços declarativos, o executor de serviços aciona o processador de consultas do sítio, que tem acesso direto ao repositório local. Caso seja necessária alguma materialização de dados AXML, o executor de serviços solicita ao avaliador AXML que invoque as chamadas de serviços. Por sua vez, o avaliador AXML pode aplicar algumas otimizações, como verificar os tipos de dados esperados de um serviço Web e aplicar seleções nos documentos AXML, visando evitar invocações desnecessárias. Por isso, o avaliador AXML também tem acesso ao processador de consultas. Tanto o avaliador AXML como o executor de serviços realizam processamento multi-tarefa.

Embora omitidos na Figura 7, vale ressaltar que existem outros recursos acessórios em um sítio *ActiveXML*, como um serviço disparador automático de

invocações programadas. O acionamento periódico de chamadas de serviços pode ser especificado no atributo "frequency".

A plataforma *ActiveXML* foi desenvolvida em Java, baseada no servidor Web Apache Tomcat (TOMCAT, 1999). Foram utilizadas bibliotecas públicas para a manipulação de dados XML, como a Apache Xerces (XERCES, 1999) e a Apache Xalan (XALAN, 1999), para apoio ao uso de padrões como o DOM (*Documento Object Model*) e as linguagens XPath e XSL-T. Também foram adotados padrões referentes à tecnologia de serviços Web, com o uso da biblioteca Apache Axis (AXIS, 2000).

Observe que um sítio *ActiveXML* pode prover outros tipos de serviços Web. Para isso, é preciso estender a interface de serviços Java do sítio com os componentes a serem publicados e configurar o servidor Web para incluir o novo serviço. Aliás, o servidor Web utilizado como infra-estrutura do sítio também oferece vários recursos para a publicação de serviços Web. Entretanto, essas alternativas ou envolvem alteração do código do sítio, ou são independentes do sistema *ActiveXML*. Por isso, não estão contempladas na Figura 7.

2.4.2 Propostas de Otimização da Materialização AXML

A materialização de um documento AXML pode ser explicitamente solicitada pelo usuário ou implicitamente disparada por consultas sobre o conteúdo do documento. Pressupõe-se que o processo de materialização sempre é iniciado no sítio-mestre, ou seja, onde o documento AXML está armazenado. Como várias alternativas de avaliação podem ser utilizadas na materialização de um documento AXML, a otimização desse processo constitui um problema complexo, com diversos aspectos a serem tratados.

Atualmente, os sítios *ActiveXML* não avaliam o desempenho das alternativas de materialização decorrentes da execução de serviços flexíveis (com nomes abstratos de serviços Web) e da delegação de tarefas. Os principais problemas de otimização tratados estão relacionados: à correta transformação de tipos de dados na troca de documentos AXML (MILO *et al.*, 2003); à replicação de serviços Web e dados em documentos AXML (ABITEBOUL *et al.*, 2003b); ao processamento de consultas com predicados de seleção sobre documentos AXML (ABITEBOUL *et al.*, 2004b); e à

otimização algébrica da materialização AXML (ABITEBOUL *et al.*, 2006a). Esses trabalhos são discutidos a seguir.

2.4.2.1 TROCA DE DOCUMENTOS AXML

Quando um documento AXML é solicitado a um sítio *ActiveXML*, geralmente é necessário enviar o conteúdo materializado do documento, mesmo que seja parcial. Vários fatores podem determinar a necessidade de materialização AXML, como segurança, custos e a capacidade do receptor de invocar chamadas de serviços Web. Assim, na troca de documentos AXML, esquemas de tipos de dados XML podem ser usados para especificar o formato esperado pelo receptor. O problema consiste em determinar se é possível transformar um documento AXML no tipo esperado e, sendo possível, quais chamadas de serviços devem ser invocadas para atingir o esquema desejado.

MILO *et al.* (2003) apresentam um algoritmo para resolver esse problema, considerando que cada invocação de chamada de serviço corresponde a uma transformação do respectivo documento AXML. Deste modo, no processo de materialização AXML, deve-se garantir que as transformações sofridas pelo documento levem a tipos de dados válidos. São propostos algoritmos baseados em autômatos para controlar essas transformações, permitindo dois níveis de garantia. Na garantia conservadora, o algoritmo de verificação é capaz de indicar se há uma seqüência segura de transformações para o documento AXML. Uma propriedade interessante do algoritmo é que nenhuma chamada de serviço precisa ser invocada. Por outro lado, se a verificação depende de determinados resultados das chamadas de serviços, o algoritmo indica quando a materialização do documento AXML possivelmente atingirá o esquema desejado.

A estratégia proposta por MILO *et al.* (2003) pode ser usada para a colaboração entre sítios *ActiveXML* na materialização de documentos AXML. Porém, essa estratégia requer que o usuário determine os sítios envolvidos, bem como os esquemas intermediários a serem utilizados durante a materialização. Por exemplo, para materializar a chamada de serviço sc9 da Figura 2, o usuário deve especificar o esquema esperado para o resultado, explicitando que P_1 aceita apenas dados regulares XML como resultado de sc9. Então, o usuário deve solicitar ao sítio P_1 a invocação de

sc9 em P_4 (ou em P_5). Isto é, neste caso, o sítio P_4 (ou P_5) deverá invocar tanto sc9 como sc10, enviando para P_1 apenas o resultado de sc9. Observe que cabe ao usuário determinar exatamente quais sítios participarão no processo de materialização e como eles devem trocar dados AXML. De fato, MILO *et al.* (2003) ressaltam a necessidade de elaborar métricas de custos e definir uma estratégia de otimização baseada em desempenho para gerar os planos de materialização.

2.4.2.2 Replicação de Serviços Web

Por ser uma mídia leve para representar informações distribuídas, documentos AXML tornam-se úteis para aplicações em dispositivos portáteis, como telefones celulares e agendas eletrônicas. Neste cenário, muitas vezes há falhas de comunicação entre pontos da rede, tornando-se interessante replicar alguns dados adequadamente. Em particular, no caso de serviços Web declarativos, é possível usar as consultas que definem esses serviços para selecionar os dados a serem replicados. Assim, pode-se replicar também a especificação do serviço Web.

ABITEBOUL *et al.* (2003b) apresentam um algoritmo baseado em um modelo de custo para escolher dados e serviços a serem replicados, considerando um determinado conjunto de documentos AXML e serviços declarativos distribuídos em um sistema P2P. Embora considere dependências entre chamadas de serviços, o modelo de custos proposto em ABITEBOUL *et al.* (2003b) não trata o impacto da ordem de invocação das chamadas, nem a carga de tarefas atribuídas aos sítios. Ou seja, o desempenho é avaliado pelo custo total da materialização. Todavia, o agendamento de tarefas tem tradicionalmente um forte impacto no desempenho da execução de *workflows* distribuídos. Além disso, devido às possibilidades de execução paralela, o tempo de resposta se torna uma métrica mais adequada. Portanto, estas são restrições importantes.

As métricas de desempenho apresentadas por ABITEBOUL *et al.* (2003b) não abrangem custos básicos relevantes na comunicação com serviços Web, como o empacotamento de mensagens SOAP. Também são ignorados custos da delegação de tarefas (*i.e.*, para enviar um subplano de materialização e coletar os resultados), que podem se tornar significativos em um documento AXML. Por fim, os autores não apresentam uma avaliação empírica das técnicas propostas.

2.4.2.3 PROCESSAMENTO DE CONSULTAS EM DOCUMENTOS AXML

No processamento de consultas sobre documentos AXML, predicados de seleção podem ser utilizados para evitar invocações desnecessárias de chamadas de serviços, como proposto em ABITEBOUL *et al.* (2004b). Neste caso, o objetivo é materializar apenas os nodos intencionais que contribuem para o resultado de uma consulta. O principal problema tratado é a identificação das chamadas de serviços cujos resultados são demandados pela consulta submetida. Algumas otimizações específicas da plataforma *ActiveXML* são propostas. Por exemplo, quando serviços declarativos são utilizados, alguns dos predicados de seleção da consulta podem ser "empurrados" para o provedor do serviço Web. Outra técnica proposta utiliza os tipos de dados esperados nos resultados de serviços Web para dispensar a invocação de certas chamadas de serviços.

Basicamente, as técnicas de otimização apresentadas em ABITEBOUL *et al.*, (2004b) focam na filtragem de chamadas de serviços a serem materializadas para satisfazer uma consulta sobre um documento AXML. Ou seja, elas não determinam como planos de materialização devem ser gerados, nem métricas de desempenho para as alternativas de avaliação. A única forma de delegação considerada é a de predicados de seleção. Além disso, elas não consideram referências abstratas para serviços Web.

2.4.2.4 OTIMIZAÇÃO ALGÉBRICA DA MATERIALIZAÇÃO AXML

Baseados nas idéias lançadas em RUBERG *et al.* (2004a), ABITEBOUL *et al.* (2006a) propõem uma álgebra para otimizar o processamento de consultas sobre o conteúdo de documentos AXML. Nessa álgebra, um documento AXML é materializado por sucessivas aplicações de regras de transformação sobre as chamadas de serviços. Tais regras são derivadas de alternativas equivalentes de materialização, abrangendo a possibilidade de delegações de tarefas e comunicação assíncrona.

Entretanto, ABITEBOUL *et al.* (2006a) não apresentam métricas de desempenho para selecionar as melhores seqüências de transformações para um documento AXML. Além disso, não é mencionada uma abordagem sistemática para a geração dessas transformações, considerando a complexidade da busca por alternativas eficientes. Também é ignorado o dinamismo dos sistemas P2P. Por fim, a álgebra proposta não contempla operadores para a descentralização do processo de otimização,

nem trata de aspectos relevantes da materialização AXML, como a localização postergada de provedores de serviços Web.

Um ponto positivo da álgebra proposta por ABITEBOUL *et al.* (2006a) consiste em regras para otimizar consultas sobre documentos AXML, incluindo a possibilidade de passar predicados de seleção para serviços declarativos. Neste ponto, essa proposta é complementar à abordagem de otimização apresentada nesta tese.

2.4.2.5 Considerações Finais

Os trabalhos apresentados sobre a otimização da materialização de documentos AXML são complementares à proposta desta tese, pois pressupõem a existência de métricas de desempenho para analisar possíveis estratégias de materialização de documentos AXML. Além disso, nenhum deles aborda aspectos práticos cruciais, como o dinamismo de sistemas P2P, a complexidade na busca por alternativas eficientes de materialização e a descentralização da otimização.

Vale ressaltar ainda que o projeto *ActiveXML* tem sido conduzido em várias frentes de pesquisa, abordando diferentes assuntos, como a análise de complexidade do processamento de consultas em documentos AXML (ABITEBOUL *et al.*, 2004d), o tratamento de eventos em redes de sensores (ABITEBOUL *et al.*, 2005a), segurança (CANAUD *et al.*, 2004), mobilidade de código (TAROPA *et al.*, 2005), índices distribuídos para sistemas P2P (ABITEBOUL *et al.*, 2004e) e controle transacional (BISWAS e KIM, 2007).

Em muitos aspectos, a materialização de documentos AXML pode ser caracterizada como a execução de *workflows* de serviços Web em um ambiente descentralizado e altamente dinâmico. Todavia, além da dificuldade inerente dos *workflows* distribuídos, materializar documentos AXML envolve complicadores adicionais, como o tratamento de respostas ativas. No próximo capítulo, é discutido o estado arte de técnicas de otimização para a execução de *workflows* distribuídos. Também são apresentados trabalhos relacionados à modelagem de desempenho da invocação de chamadas de serviços Web e da carga de processamento dos sítios em um sistema P2P.

Capítulo 3

WORKFLOWS DISTRIBUÍDOS:

ESTADO DA ARTE

Neste capítulo, são discutidas técnicas para a execução eficiente de workflows em ambientes distribuídos e heterogêneos. É apresentado um estudo sobre o estado da arte dos principais otimizadores da literatura, com ênfase em abordagens dinâmicas e descentralizadas. Também são abordados trabalhos sobre a estimativa de custos no uso de serviços Web.

3.1 INTRODUÇÃO

A otimização da execução de processos inter-relacionados em ambientes distribuídos, heterogênos e com controle descentralizado envolve uma vasta gama de problemas, tocando diversas áreas de pesquisa. Em especial, muitos avanços têm sido registrados em sistemas para gerência de *workflows* na Web. Contudo, as soluções propostas ainda não respondem as principais questões que surgem em cenários complexos como os sistemas P2P e os *grids*.

Em linhas gerais, um *workflow* consiste na automação de um processo de negócio, no todo ou em parte, durante o qual documentos, informações e/ou tarefas são passados de um participante para outro para executar ações conforme um conjunto de regras procedimentais (WFMC, 1999). Nesse contexto, existe uma distinção fundamental entre a especificação de um *workflow* e sua execução. Basicamente, a especificação de um *workflow* descreve um conjunto de tarefas e seus relacionamentos, além de informações sobre participantes e critérios de execução. Ela pode ser concreta ou abstrata, caso seja diretamente exeqüível ou não, respectivamente (YU e BUYYA, 2005). Isto é, uma especificação concreta inclui a seleção de todos os recursos físicos necessários para a execução do *workflow*. Por outro lado, especificações abstratas são mais flexíveis e permitem execuções mais facilmente adaptáveis (HAN *et al.*, 1998, RUSSELL *et al.*, 2004b, VIEIRA, 2005).

A execução de um *workflow* corresponde à realização de sua especificação em um determinado sistema, pela qual são produzidos resultados. Cada execução representa uma instância do *workflow*. Um sistema para gerência de *workflows* (SGW) oferece mecanismos para especificar, executar e monitorar o andamento das tarefas de um *workflow* em um conjunto de recursos computacionais. Embora tenham se destacado em ambientes corporativos, esses sistemas estão se tornarando cada vez mais úteis em diversas áreas, como, por exemplo, no apoio à realização de experimentos científicos (MEYER, 2005). Essa popularidade tem sido reforçada com o uso de padrões relacionados à tecnologia de serviços Web.

A execução eficiente de um *workflow* é geralmente guiada por um plano de avaliação, que descreve tanto os recursos a serem utilizados como a ordem de

acionamento das tarefas, a partir da especificação do *workflow* (BLYTHE *et al.*, 2005, DEELMAN *et al.*, 2003, KWOK e AHMAD, 1999). A geração de planos de avaliação pode se tornar particularmente difícil para especificações abstratas, pois é preciso antes traduzir as referências abstratas e identificar os recursos necessários (BLYTHE *et al.*, 2003). Em alguns casos, essa tradução também pode acrescentar novas tarefas à especificação. Por exemplo, isso ocorre quando uma tarefa abstrata é implementada por um conjunto de tarefas físicas. A partir de uma especificação de *workflow* (seja abstrata ou concreta) pode ser derivado um conjunto de planos equivalentes, sendo necessário estabelecer critérios para selecionar as melhores soluções. Esses critérios costumam ser baseados em métricas de qualidade de serviço (ou QoS, de "*Quality of Service*"), como desempenho, confiabilidade e disponibilidade (AZEVEDO, 2003).

Devido à crescente necessidade de integração e colaboração dos sistemas atuais, tem sido bastante explorada a execução de *workflows* em ambientes distribuídos (ARORA *et al.*, 2002, BENATALLAH *et al.*, 2005, CAO *et al.*, 2003, MEYER *et al.*, 2006a, VARGAS *et al.*, 2005, VIEIRA, 2005, YU e BUYYA, 2005). Como na execução de *workflows* em sistemas P2P e *grids*, na materialização de documentos AXML são observados os seguintes aspectos:

- diferentes padrões de fluxo de controle, definidos pela especificação de relacionamentos entre chamadas de serviços Web;
- dependências de invocação, resultando em restrições na ordem de invocação de chamadas de serviços Web e em fluxos de dados durante a materialização de um documento AXML;
- execução flexível de serviços Web, devido ao uso de endereços abstratos que podem ser mapeados em endereços físicos equivalentes, possivelmente providos por diferentes sítios;
- cenários dinâmicos, afetados tanto por variações de desempenho dos recursos envolvidos como por adesões e saídas aleatórias de participantes do sistema; e
- delegação de tarefas, quando outros sítios do sistema colaboram com o sítio mestre na invocação de chamadas de serviços, permitindo a descentralização do processo de materialização.

Vale ressaltar que, como a materialização de documentos AXML ocorre em sistemas P2P, problemas tradicionalmente enfrentados na execução de *workflows* distribuídos são potencializados pela grande escala, alta volatilidade dos sítios e descentralização de controle. Uma caracterização detalhada dos sistemas P2P pode ser encontrada em MILOJICIC *et al.* (2002). Entre os principais problemas enfrentados, destacam-se a falta de conhecimento global sobre os participantes do sistema (especialmente da carga de processamento dos sítios) e a alta complexidade na geração de planos de avaliação eficientes, devido ao grande número de variáveis a serem consideradas na otimização. Deve-se levar em conta a heterogeneidade tanto de desempenho dos sítios como dos enlaces de comunicação, além das diferentes políticas de gerência de recursos. Existe também a dificuldade em garantir propriedades importantes, como finalização e convergência da execução distribuída.

Por outro lado, a materialização de documentos AXML se diferencia da execução de *workflows* distribuídos em alguns pontos relevantes. Enquanto resultados parciais são raramente úteis em um SGW, documentos AXML podem ser materializados e consumidos incrementalmente. Assim, convém identificar as partes do documento a serem materializadas e favorecer a obtenção antecipada dos primeiros resultados. Outro diferencial da materialização AXML consiste na possibilidade de respostas ativas, que podem mudar significativamente a especificação de um documento AXML em tempo de execução. Em um SGW, alterações dinâmicas nas especificações não costumam ser consideradas. Além disso, a maioria dos otimizadores para *workflows* distribuídos realiza agendamento dinâmico considerando apenas as tarefas prontas para execução. Com isso, reduções de custos de transferências de dados são obtidas principalmente movendo as próximas tarefas para os sítios que já possuem as entradas necessárias. Todavia, os serviços Web referenciados por um documento AXML podem ser providos por máquinas que não permitem armazenar resultados intermediários, pois não implementam um sítio do sistema P2P. Ou seja, na materialização de um documento AXML, minimizar custos de transferência de dados requer agrupar corretamente as chamadas de serviços antes de invocá-las, por intermédio da delegação de tarefas.

Os componentes de um SGW tratam basicamente do projeto e da execução de *workflows* (YU e BUYYA, 2005). Em particular, componentes que implementam

funções de execução estão relacionados a atividades de planejamento, realização e monitoração de instâncias de *workflows*. Esses componentes são responsáveis por garantir o bom desempenho das instâncias de um *workflow*. Neste capítulo, documentos AXML são analisados sob a ótica da gerência de *workflows*. Na seção 3.2 são abordados os artifícios necessários para a especificação e representação de documentos AXML, comparando-os com propostas da literatura relacionadas ao projeto de *workflows*. Já na seção 3.3, são apontadas as principais características de otimizadores para a execução de *workflows*. Na seção 3.4 são discutidas propostas de otimizadores com ênfase em técnicas descentralizadas. A seção 3.5 trata de um aspecto crucial para a otimização de *workflows*, que é a estimativa de métricas de desempenho para comparar planos equivalentes. Por fim, na seção 3.6 são feitas algumas considerações sobre a otimização da materialização de documentos AXML, evidenciando os requisitos que não são atendidos pelas propostas existentes na literatura.

3.2 REPRESENTAÇÃO DE *WORKFLOWS* COMPLEXOS

As funções de apoio ao projeto de *workflows* estão relacionadas à definição de estrutura, à especificação (ou modelo) e ao método de composição usado para construir um *workflow* (YU e BUYYA, 2005). A estrutura compreende o conjunto de construtores básicos de controle e de dados que podem ser utilizados para compor um *workflow*. Exemplos desses construtores são a seqüência de tarefas e a iteração. Já a especificação de um *workflow* reúne um conjunto de definições particulares das tarefas e da estrutura do *workflow*, podendo ser concreta ou abstrata.

O método de composição de um SGW é determinado pelos mecanismos utilizados para a construção de *workflows*. A composição pode ser definida pelo usuário ou automática, quando inferida a partir de propriedades das atividades, participantes e recursos do *workflow*. Métodos orientados a usuário podem ser gráficos, com o apoio de ferramentas como no sistema Triana (TRIANA, 2003, MAJITHIA *et al.*, 2004), ou baseados em linguagens. Esses últimos são bem mais comuns em ambientes distribuídos. De fato, existem várias linguagens para a especificação de *workflows* distribuídos, como a VDL (de "*Virtual Data Language*") do sistema Chimera (FOSTER *et al.*, 2002), a linguagem de roteiros do Condor (THAIN *et al.*, 2005) para *grids* e a WS-BPEL (OASIS, 2007) para serviços Web. Aliás, em se tratando de serviços Web,

essa diversidade de linguagens é bem ampla (HULL e SU, 2005). Por outro lado, também merece destaque o padrão BPMN (de "*Business Process Modeling Notation*") (BPMN, 2004, OMG, 2006), que consiste em uma notação gráfica para descrever processos de negócio. Observe que métodos gráficos demandam mecanismos de tradução para linguagens de *workflows*, como ocorre na ferramenta de código livre proposta por CHUN *et al.* (2006), que traduz a BPMN para BPEL.

Em um documento AXML, fluxos de controle e de dados são especificados por meio de chamadas de serviços Web. Algumas linguagens também exploram a idéia de embutir chamadas de programas em arquivos XML. Este é o caso da *Microsoft* XAML (de "*eXtensible Application Markup Language*") (GRIFFITHS, 2004), cujo propósito é semelhante ao da linguagem HTML. A XAML permite especificar interfaces de usuário em arquivos XML, oferecendo vários recursos gráficos sofisticados, como a rotação de elementos e animação. Na XAML, chamadas de métodos podem ser descritos por elementos XML. Isso também está presente na linguagem *Apache Jelly* (JELLY, 2006) e na plataforma *Macromedia ColdFusion* MX (COLDFUSION, 2006). Entretanto, somente na linguagem AXML a tecnologia de serviços Web têm sido explorada efetivamente para integração de dados e programas em sistemas P2P.

Tabela 2. Padrões de fluxos de controle encontrados em documentos AXML.

Padrão	Forma de especificação
seqüência	Tanto pelo atributo "followed_by" como por aninhamento de chamadas de serviços.
divisão paralela	Compartilhamento de resultados de chamadas de serviços, por meio de parâmetros de entrada definidos por expressões XPath.
sincronização	Aninhamento de chamadas de serviços.
múltiplas instâncias sem sincronização	Pelo atributo "followed_by", pois cada referência gera uma nova invocação de chamada de serviço.

3.2.1 Padrões de Fluxos de Controle e de Dados

Em linguagens para especificação de *workflows* podem ser identificados padrões básicos de fluxos de controle (AALST *et al.*, 2003). Ou seja, cada especificação de

workflow é essencialmente formada por composições desses padrões. Em particular, um documento AXML pode conter os construtores indicados na Tabela 2. Com esses padrões, a linguagem AXML permite especificar *workflows* bastante complexos, contendo dependências de dados e restrições na ordem de invocação das chamadas de serviços dos documentos AXML. Em um documento AXML, a passagem de dados entre tarefas ocorre explicitamente por meio de parâmetros (por valor), conforme a abordagem descrita em RUSSELL *et al.* (2004a) sobre fluxos de dados em *workflows*.

Especificações de *workflows* são geralmente transformadas em uma representação interna, para então poderem ser executadas. Uma representação muito utilizada é a baseada em grafos de dependências (BLYTHE *et al.*, 2003, EROL *et al.*, 1994, HULL *et al.*, 2000, LIU *et al.*, 2004b, KWOK e AHMAD, 1999, VARGAS *et al.*, 2004), em alguns casos permitindo ciclos. Tal representação facilita a manipulação dos construtores que compõem o *workflow* e permite verificar propriedades importantes, como a existência de bloqueios perpétuos (*i.e.*, *deadlocks*). Outros formalismos também podem ser adotados, como as redes de Petri (AALST, 1998, PETERSON, 1977) e o cálculo π (MILNER *et al.*, 1992a, 1992b, MILNER, 1993), que consiste em uma álgebra de processos concorrentes para ambientes dinâmicos. Esses formalismos são baseados na teoria de autômatos e oferecem notações com grande poder de expressão, porém geralmente bastante complexas. Isso dificulta principalmente a representação de *workflows* com muitas tarefas e/ou participantes. Redes de Petri são bastante interessantes para verificar propriedades de *workflows* que contêm expressões condicionais. Contudo, essas expressões não ocorrem em documentos AXML.

Nesta tese, um documento AXML é representado por um grafo orientado, por um formalismo semelhante ao adotado em SGWs. São considerados dois tipos de relacionamentos entre chamadas de serviços: dependências e conseqüências de invocação (*i.e.*, chamadas colaterais), definidas formalmente no Capítulo 4. Embora essa distinção nem sempre seja feita em SGWs, ela é relevante para a descentralização da materialização de documentos AXML, já que é necessário dividir corretamente o grafo de dependências para poder distribuí-lo entre os sítios do sistema. Observe que enquanto a materialização de uma chamada de serviço implica necessariamente na invocação de suas dependências, ela não exerce nenhum efeito sobre as chamadas que a referenciam pelo atributo "followed_by". O formalismo proposto nesta tese inclui ainda

critérios para a simplificação de redundâncias e a verificação da correção do grafo de dependências quanto à ocorrência de ciclos e *deadlocks*.

3.2.2 Resultados Persistentes

No grafo de dependências de um documento AXML, além dos controles de fluxo e de dados, é importante indicar os resultados de chamadas de serviços que são persistentes. Isso porque em documentos AXML podem existir chamadas de serviços cujos resultados são intermediários. Ou seja, que são utilizados apenas como entrada de outras chamadas de serviços, não compondo o conteúdo do documento. Portanto, quando há delegação da invocação de chamadas de serviços na materialização de um documento AXML, esses resultados não precisam ser transferidos para o sítio mestre.

Embora não sejam diretamente referenciados na classificação de fluxo de dados proposta em RUSSEL *et al.* (2004a), resultados persistentes podem ser vistos como interações de dados entre instâncias do *workflow* e o ambiente de execução. Esse tipo de interação é representada pelo padrão 18 em RUSSEL *et al.* (2004a). Vale ressaltar que os autores afirmam que esse padrão é raramente encontrado em SGWs.

A identificação de resultados persistentes em *workflows* ocorre no sistema Vortex (HULL *et al.*, 2000). A ênfase do Vortex é o apoio à execução de fluxos de decisões, os quais consistem em *workflows* cujas transições entre tarefas são determinadas por condições pré-estabelecidas pelo usuário. Uma execução no Vortex visa determinar um conjunto de valores (ditos atributos) a serem computados por tarefas de um *workflow*, sendo alguns desses atributos temporários. O Vortex tenta otimizar a execução de *workflows* por meio de uma análise de propagação de condições sobre atributos. Uma vantagem dessa análise consiste em identificar atributos intermediários que se tornam desnecessários durante a execução de um *workflow*. Contudo, o sistema Vortex não trata da seleção de recursos para a execução de *workflows* e possui controle centralizado.

3.2.3 Evolução Dinâmica

Um requisito da representação de documentos AXML é o tratamento de respostas ativas, que ocorrem quando o resultado de uma chamada de serviço contém

outras chamadas de serviços. Assim, são necessárias técnicas para atualizar o grafo de dependências de um documento AXML durante a materialização (*i.e.*, em tempo de execução). Apesar de existirem alguns trabalhos que abordam a evolução da especificação de *workflows*, este ainda é um tema aberto para pesquisa.

A maioria dos trabalhos sobre evolução de *workflows* trata problemas relacionados à validade das especificações obtidas por modificações. Uma proposta importante nessa área foi apresentada por CASATI *et al.* (1996), que introduzem um conjunto de primitivas para atualizar especificações de *workflows* e propõem uma taxonomia para as abordagens de evolução. Essas primitivas constituem a base da linguagem WFML (de "*WorkFlow Modification Language*"). Entretanto, a abordagem proposta por CASATI *et al.* (1996) pressupõe uma estratégia gulosa de geração de planos de execução. Os autores consideram ainda que o controle da execução de *workflows* é centralizado e que o tempo gasto em otimização é irrisório. Vale ressaltar que essas premissas não são realistas em sistemas P2P.

A evolução de *workflows* também é abordada em EDER e SARINGER (2003), cuja principal preocupação é lidar com as instâncias que estão sendo executadas quando uma modificação é feita na especificação do *workflow*. Basicamente, os autores propõem uma estratégia para migrar as instâncias correntes (com a especificação antiga) para uma nova especificação, por meio da identificação das tarefas que satisfazem essa nova especificação e das tarefas que precisam ser compensadas. Já KRADOLFER e GEPPERT (1999) propõem uma abordagem baseada em invariantes para derivar as condições de migração de uma instância de *workflow*. Essas invariantes são propriedades que devem ser satisfeitas antes e após qualquer modificação na especificação de um *workflow*. Em ZHANG *et al.* (2005), além da migração das instâncias correntes de um *workflow*, considera-se também a reconfiguração de recursos utilizados. Porém, essa reconfiguração não descreve os critérios necessários para selecionar os melhores recursos para cada nova tarefa.

Em documentos AXML, respostas ativas alteram a especificação do problema sempre com o acréscimo de novas tarefas. Esse tipo de alteração não implica em compensação de tarefas, mas costuma demandar a criação de relacionamentos com tarefas pré-existentes. Como tais relacionamentos podem ser numerosos, nesta tese são apresentados mecanismos para simplificar a modificação do grafo de dependências.

Além disso, para as chamadas de serviços embutidas em um documento AXML, os parâmetros de entrada definidos por expressões XPath precisam ser re-avaliados, disparando operações que geralmente possuem um alto custo. Assim, pode ser interessante postergar essa re-avaliação por um período determinado. No Capítulo 4, são descritas estruturas para controlar as modificações resultantes de respostas ativas durante a materialização de um documento AXML. Não foram encontradas propostas em SGW que tratem esses aspectos na evolução de *workflows*.

KAMMER *et al.* (2000) apontam que um aspecto fundamental para apoiar a evolução de *workflows* consiste em mecanismos que permitam execuções parciais. No sistema D2WMS (DIAS *et al.*, 2003), execuções parciais de *workflows* são exploradas em ambientes com desconexão de rede. O D2WMS também descreve controles para descentralizar a execução de um *workflow* entre sítios. Similarmente, nesta tese, o otimizador XCraft conduz a materialização de documentos AXML por meio de execuções parciais. Todavia, ao contrário do XCraft, no D2WMS pressupõe-se que sub-*workflows* são determinados pelo usuário no momento da especificação.

Outro ponto relevante na evolução de *workflows* é o impacto das mudanças em planos previamente gerados. Decisões anteriores podem se tornar inválidas com a nova especificação, demandando uma revisão do planejamento feito pelo otimizador. Em muitos casos, os ajustes necessários para garantir uma execução eficiente podem atingir todo o plano. Assim, convém restringir as áreas que serão afetadas pelas mudanças. A importância de limitar processos de re-otimização a partes de um plano já foi observada em outros contextos, como destacado por BABU e BIZARRO (2005) sobre técnicas adaptativas para o processamento de consultas.

Na execução de *workflows*, SAKELLARIOU e ZHAO (2004b) mostram que técnicas de re-otimização seletiva geralmente são bem mais efetivas. Os autores propõem que a re-otimização de planos gerados estaticamente seja restrita a apenas algumas tarefas. Para selecionar essas tarefas, é estimado o tempo de "folga" de cada tarefa, considerado como o tempo máximo de atraso permitido em relação ao término do *workflow*. Contudo, essa solução envolve um alto custo no cálculo e manutenção dessas estatísticas em ambientes dinâmicos. Por outro lado, estratégias dinâmicas de geração de planos costumam apoiar mais facilmente a aplicação seletiva de re-otimizações. Essa abordagem é explorada no XCraft.

Técnicas de re-otimização dinâmica também são interessantes para tratar a ocorrência de falhas, como adotado no sistema METEOR-S (VERMA *et al.*, 2005). Todavia, em geral trabalhos sobre re-otimização da execução de *workflows* consideram uma arquitetura centralizada. Por outro lado, em ARORA *et al.* (2002), protocolos de comunicação P2P promovem um balanceamento de carga dinâmico e descentralizado. Porém, o foco da proposta é apoiar a adaptação da execução distribuída de um *workflow* em relação a mudanças de configuração do sistema. Ou seja, não são consideradas mudanças de especificação. Além disso, ARORA *et al.* (2002) pressupõem que todas as tarefas executadas são independentes. No XCraft, sítios que colaboram entre si podem monitorar a execução sub-planos e decidir re-otimizá-los quando for conveniente.

3.3 PLANEJAMENTO DA EXECUÇÃO DE *WORKFLOWS*

Basicamente, o planejamento da execução distribuída de um *workflow* engloba duas tarefas principais: a seleção de recursos e o agendamento da execução das tarefas. Na seleção de recursos, deve-se identificar os sítios candidatos a executar e/ou controlar cada tarefa. Em seguida, os candidatos devem ser classificados por algum critério, para que então sejam selecionados os sítios que participarão da execução do *workflow*. Já no agendamento da execução das tarefas, o objetivo é determinar uma ordem de avaliação que minimize o tempo de execução do *workflow*, explorando o paralelismo entre tarefas quando for possível. Essas duas atividades exercem influência mútua, pois a seleção de recursos determina as seqüências de tarefas que rodam em um mesmo sítio e o agendamento da execução afeta a carga de processamento dos sítios.

Mesmo em ambientes estáveis, um desafio importante da otimização de *workflows* distribuídos é a geração e comparação de planos de avaliação. Quando se considera a descentralização da execução, ótimos locais não garatem ótimos globais. Ou seja, a substituição de um subplano por um outro equivalente de custo inferior pode produzir um plano mais caro. Assim, apenas planos completos podem ser comparados para determinar uma solução ótima global. Portanto, neste contexto não podem ser explorados métodos de busca clássicos (RUSSELL e NORVIG, 2003), como a programação dinâmica e a poda (mais conhecido como "*Branch & Bound*"). Geralmente, esses métodos são baseados na monotonicidade de custo na composição de subplanos, como explorado em DOAN e HALEVY (2002) para ordenar planos de

consultas em sistemas de integração de informações. Tal propriedade não ocorre em ambientes descentralizados, particularmente na materialização de documentos AXML.

De fato, por ser um problema tradicionalmente reconhecido como NP-completo (KWOK e AHMAD, 1999, RUSSELL e NORVIG, 2003), o planejamento da execução de *workflows* distribuídos somente consegue garantir soluções ótimas em cenários muito restritos. Por exemplo, em SRIVASTAVA *et al.* (2006) são propostos algoritmos para gerar planos ótimos para consultas que recuperam dados a partir de serviços Web, em uma abordagem semelhante à execução de *workflows*. Entretanto, os planos gerados sempre correspondem a florestas, sem a ocorrência de divisões paralelas ou dependências compartilhadas. Além disso, esses algoritmos enfocam apenas o agendamento da execução de tarefas, ignorando a seleção de recursos. Pressupõem-se recursos infinitos, tal que cada serviço é executado em um sítio distinto, ignorando a carga de processamento dos sítios.

3.3.1 Uso de Heurísticas

Face à natureza combinatória da otimização da execução de *workflows* distribuídos, várias heurísticas têm sido propostas (BRAUN *et al.*, 2001, DONG e AKL, 2006, KWOK e AHMAD, 1999). Por exemplo, em *grids*, são amplamente utilizadas a estratégia oportunista de balanceamento de carga (conhecida como OLB, de "*Opportunistic Load Balancing*") e a de menor tempo de execução (ou MET, de "*Minimum Execution Time*"). Contudo, essas heurísticas geralmente simplificam demasiadamente o problema. Por exemplo, na heurística MET, apenas a capacidade de processamento dos sítios é considerada, ignorando-se a carga de tarefas. Ou seja, ela tende a sobrecarregar sítios com bom desempenho. Já na OLB, apenas a carga de tarefas dos sítios é avaliada, podendo mapear tarefas pesadas em sítios com baixa capacidade de processamento. Além disso, essas heurísticas visam principalmente *workflows* com tarefas independentes entre si.

Para *workflows* com tarefas relacionadas, um algoritmo bastante conhecido é o HEFT (de "*Heterogeneous Earliest Finish Time*") (DONG e AKL, 2006), que atribui pesos a tarefas e relacionamentos antes de determinar um mapeamento em sítios. Porém, o HEFT considera uma aproximação (*e.g.*, média, mediana, melhor tempo, etc.) do tempo de execução das tarefas em diferentes sítios, a qual pode apresentar desvios

significativos de desempenho conforme o método escolhido. Meta-heurísticas também podem ser usadas para o planejamento da execução de *workflows*, como é o caso dos algoritmos genéticos (RUSSELL e NORVIG, 2003) e da GRASP (de "*Greedy Random Adaptive Search Procedure*") (FEO e RESENDE, 1995). Por exemplo, no otimizador ASKALON (WIECZOREK *et al.*, 2005), foram implementados o algoritmo HEFT e um algoritmo genético. Em uma análise empírica, WIECZOREK *et al.* (2005) observaram que os algoritmos genéticos demandaram significativamente mais tempo de otimização do que o HEFT no ASKALON. A meta-heurística GRASP utiliza uma busca local estocástica, pela qual soluções gulosas são aleatoriamente modificadas na tentativa de encontrar soluções melhores. A GRASP é interessante em ambientes dinâmicos, porém suas aplicações não lidam com a possibilidade de delegação de tarefas (BLYTHE *et al.*, 2005, WU *et al.*, 2001).

Nesta tese, são propostos um algoritmo dinâmico e uma álgebra de operadores que permitem a otimização incremental de planos de materialização, considerando a colaboração entre sítios. Em cada passo da otimização, o XCraft pode escolher entre diferentes alternativas para produzir subplanos físicos de materialização, como a geração exaustiva e o uso da heurística gulosa. Para superar as limitações dessas alternativas, em PEREIRA, RUBERG e MATTOSO (2006) foram exploradas técnicas de busca local para auxiliar a geração e comparação de planos de materialização de documentos AXML, por meio da estratégia SLS-MC (de "*Stochastic Local Search with Multiple stop Conditions*"). O cerne da SLS-MC é uma meta-heurística estocástica que refina sucessivamente uma solução inicial até que certas condições de parada sejam satisfeitas. Ao longo da busca, planos piores podem ser aleatoriamente aceitos, visando fugir de mínimos locais. São consideradas as dependências entre chamadas de serviços e a delegação de tarefas.

Uma vantagem relevante da SLS-MC consiste no controle do tempo gasto na otimização, pela especificação de diferentes condições de parada. Além disso, a SLS-MC pode ser configurada com diferentes heurísticas para agendamento da execução de tarefas.

3.3.2 CLASSIFICAÇÃO DOS OTIMIZADORES

Um SGW geralmente possui um componente otimizador encarregado de planejar as execuções de *workflows*. O comportamento desse componente é determinado por vários aspectos, conforme apresentado na taxonomia proposta por YU e BUYYA (2005) para sistemas de *workflows* em *grids*. Entre esses aspectos, destacam-se a abrangência das decisões, o momento da otimização e a arquitetura de controle, os quais influenciam fortemente a geração de planos de avaliação. A Figura 8 mostra a principal parte da taxonomia de YU e BUYYA (2005) relativa ao planejamento da execução de *workflows*.

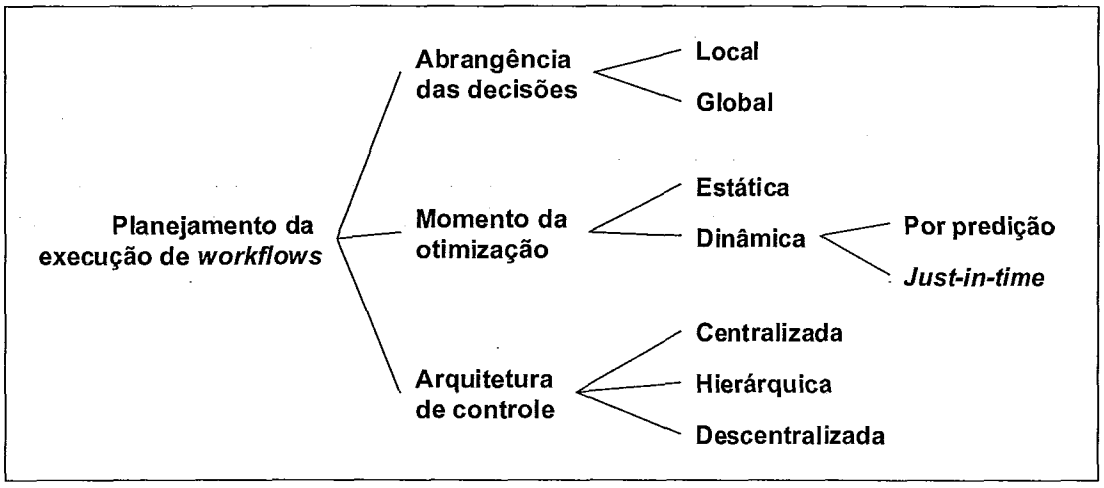


Figura 8. Principais características de otimizadores para a execução de *workflows* distribuídos, baseadas na taxonomia de YU e BUYYA (2005).

3.3.2.1 ABRANGÊNCIA DAS DECISÕES

Quanto à abrangência das decisões, o planejamento de um *workflow* pode ser local, se as tarefas são analisadas individualmente, ou global, caso considere o desempenho das tarefas conjuntamente. Em um ambiente distribuído, a otimização global deve levar em conta o volume de dados transferidos entre tarefas e a possibilidade de execução paralela. Decisões locais são baseadas em estratégias gulosas e costumam ser míopes para esses fatores, os quais têm forte impacto no desempenho da execução de um *workflow*. Vale lembrar que essas estratégias podem ser efetivas em cenários mais simples. Por exemplo, em uma rede de sensores, ABRAMS e LIU (2006) apresentam resultados interessantes obtidos com um algoritmo guloso. Todavia, os

autores não consideram o tempo de execução das tarefas nos diferentes sítios e o controle da execução do *workflow* é centralizado.

Vários otimizadores para *workflows* são baseados em algoritmos com decisões locais (ARORA *et al.*, 2002, BENATALLAH *et al.*, 2005, DAGMAN, 2006, FOSTER, 2002, 2003, HUANG *et al.*, 2005, OINN *et al.*, 2004, THAIN *et al.*, 2005, TRIANA, 2003, TAYLOR *et al.*, 2003). Uma boa parte desses trabalhos pressupõe que os *workflows* são do tipo "*bag-of-tasks*", possuindo somente tarefas independentes entre si. Porém, uma parcela substancial de aplicações de *workflows* explora relacionamentos entre tarefas (AALST *et al.*, 2003). Nesses casos, quando são consideradas transferências de dados, BLYTHE *et al.* (2005) evidenciam em uma análise experimental que um algoritmo local amiúde apresenta desempenho significativamente inferior ao de um algoritmo global.

Embora decisões globais produzam planos mais eficientes, como o planejamento da execução de *workflows* distribuídos envolve uma enorme complexidade, avaliar planos completos é impraticável na maioria dos casos. Convém determinar uma forma adequada de divisão e geração incremental de planos, ao mesmo tempo considerando fatores relevantes e mantendo a complexidade sob controle. Por isso, o planejamento global de *workflows* distribuídos está quase sempre associado a estratégias sub-ótimas, como em BLYTHE *et al.* (2005), GOUNARIS *et al.* (2004b), WIECZOREK *et al.* (2005) e em WU *et al.* (2001).

3.3.2.2 MOMENTO DA OTIMIZAÇÃO

O momento em que ocorre a otimização determina se a geração de planos é estática ou dinâmica. Em uma abordagem estática, os planos completos são produzidos antes da execução, considerando as informações disponíveis no momento da geração. Em BLYTHE *et al.*, (2004), pressupõe-se que as execuções de *workflows* são longas, logo a etapa de otimização pode demorar. Também considera um ambiente mais estável.

Em ambientes com participantes autônomos e distribuídos em redes heterogêneas, é comum ocorrerem variações de carga e desempenho ao longo de períodos de funcionamento. Nestes casos, um plano gerado estaticamente pode facilmente se tornar desatualizado. Esse efeito é ainda pior em sistemas P2P, pois sítios

entram e saem do sistema arbitrariamente, podendo invalidar os planos gerados *a priori*. Apesar dessas limitações, a geração estática de planos ainda é bastante comum em otimizadores para a execução de *workflows* distribuídos (ALHUSAINI *et al.*, 1999, BAUER e DADAM, 1997, BLYTHE *et al.*, 2005, BRAUN *et al.*, 2001, CAO *et al.*, 2003, GOUNARIS *et al.*, 2004b, VARGAS *et al.*, 2005, WIECZOREK *et al.*, 2005, ZENG *et al.*, 2004, ZHANG *et al.*, 2006).

Por outro lado, a geração dinâmica de planos se baseia em iterações (ditas passos) de otimização e execução, levando em conta tanto informações estáticas como dinâmicas. Isso permite ao otimizador ajustar decisões tomadas previamente, além de postergar determinadas escolhas para um momento mais adequado (*i.e.*, quando houver informações mais atualizadas sobre o sistema). De fato, sistemas distribuídos e dinâmicos requerem adaptatividade em diferentes níveis (GOUNARIS *et al.*, 2004a, HAN *et al.*, 2004), abrangendo desde a identificação de recursos para a geração de planos até a execução de tarefas.

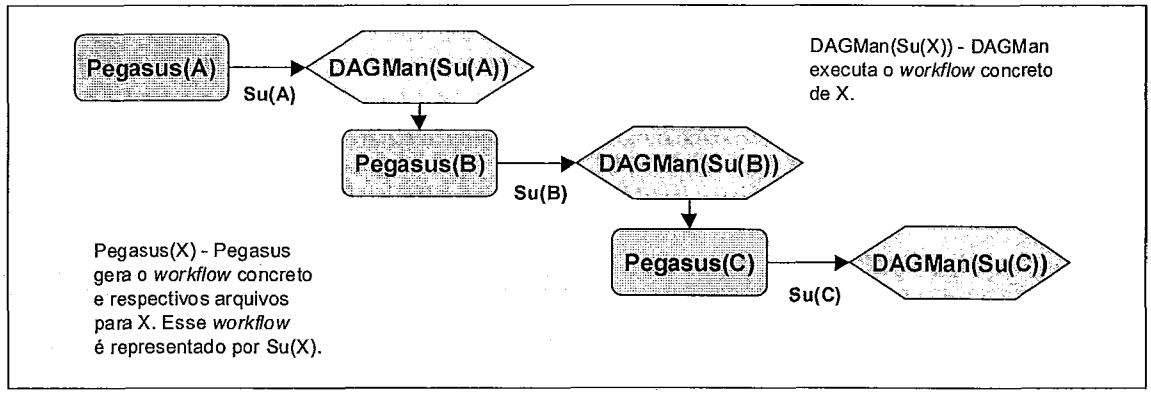


Figura 9. Planejamento dinâmico "*just-in-time*" no otimizador Pegasus (DEELMAN *et al.*, 2004).

Passos de planejamento dinâmico podem ser observados no Pegasus (DEELMAN *et al.*, 2004, 2005), conforme ilustrado na Figura 9. Esse otimizador é baseado em dois componentes principais: um gerador de *workflow* abstrato, para identificar os serviços necessários para a composição do *workflow* a partir de uma descrição de alto nível; e um gerador de *workflow* concreto, que seleciona os recursos a serem utilizados na execução. O Pegasus quebra os planos em partes menores e realiza um planejamento dinâmico para cada parte. Porém, essa divisão de cada plano é bem simples, baseada somente no nível dos tarefas do grafo de dependências, tal que tarefas

em um mesmo nível são agrupadas. Para a execução de cada sub-*workflow*, o Pegasus utiliza o DAGMan (DAGMAN, 2006, THAIN *et al.*, 2005).

Também existem estratégias híbridas, que tentam reduzir a complexidade da otimização quando são considerados conjuntamente: a seleção de recursos e o agendamento da execução de tarefas. Por exemplo, na arquitetura de *grid* virtual inicialmente proposta por KEE *et al.* (2005), especificações de *workflows* utilizam recursos descritos pela linguagem *vgDL* (de "*virtual grid Description Language*"). Nessa linguagem, recursos são classificados e agrupados conforme critérios definidos pelo usuário. Para a execução de um *workflow*, a seleção de recursos é estática e baseada em decisões locais, sendo feita pelo componente *vgFAB* (de "*virtual grid Finder And Binder*"). Porém, ZHANG *et al.* (2006) estendem essa estratégia, tornando dinâmico o agendamento da execução de tarefas. Os autores propõem a pré-seleção estática de recursos visando diminuir o tempo de geração de planos.

O planejamento dinâmico de *workflows* pode ser baseado em predição de desempenho ou orientado a tarefas prontas para execução, também conhecido como "*just-in-time*". Este última é muito utilizado por algoritmos gulosos ou oportunistas, como no Pegasus e no escalonador de tarefas Condor (SINGH *et al.*, 2006). Entretanto, em testes experimentais, JANG *et al.* (2005) verificaram que a seleção de recursos baseada em um modelo de predição geralmente é mais eficiente que a seleção "*just-in-time*" baseada em carga ou capacidade dos sítios. No XCraft, é adotada uma estratégia dinâmica baseada em um modelo de custo que estimo o desempenho de diferentes alternativas de planos de materialização.

3.3.2.3 ARQUITETURA DE CONTROLE

Conforme o tipo de controle usado no planejamento e na execução de *workflows*, a arquitetura do otimizador pode ser centralizada, hierárquica ou descentralizada. Essas duas últimas arquiteturas são mais escaláveis, porém envolvem uma maior complexidade. Um otimizador centralizado concentra o controle do planejamento e da execução em um único sítio. Já em arquiteturas hierárquicas, adota-se uma estratégia híbrida, baseada em um otimizador central que distribui seja o planejamento ou o controle da execução para outros otimizadores de mais baixo nível. Neste caso, os otimizadores de baixo nível são limitados e não podem solicitar a

colaboração de outros sítios. Além disso, geralmente o controle da execução das tarefas é realizado pelo otimizador central, que produz todo o plano de avaliação, deixando apenas a execução para os otimizadores de baixo de nível, como em VARGAS *et al.* (2005). Esse papel central não existe em uma arquitetura descentralizada, na qual vários otimizadores podem colaborar tanto para o planejamento como para a execução de um *workflow*.

Em se tratando de serviços Web, arquiteturas para planejamento e execução de *workflows* com serviços Web estão relacionadas aos conceitos de orquestração ou coreografia (BARROS *et al.*, 2005). Orquestração pressupõe que há algum controlador que aciona os serviços Web envolvidos, enquanto coreografia se aplica a serviços que interagem entre si para negociar a colaboração. Na materialização de documentos AXML, são necessárias tanto técnicas de orquestração de serviços Web, para invocar as chamadas de serviços embutidas nos documentos, como de coreografia, para permitir a colaboração entre os sítios do sistema durante a materialização.

Embora arquiteturas descentralizadas sejam mais adequadas para a grande escala e o dinamismo inerentes de *grids* e sistemas P2P, a maioria dos otimizadores para *workflows* distribuídos é centralizada (ALHUSAINI *et al.*, 1999, BLYTHE *et al.*, 2004, 2005, DASILVA *et al.*, 2003, DEELMAN *et al.*, 2003, 2004, FOSTER *et al.*, 2002, GOUNARIS *et al.*, 2004b, HUANG *et al.*, 2005, PEGASUS, 2006, SAKELLARIOU e ZHAO, 2004a, SINGH *et al.*, 2006, VERMA *et al.*, 2005, WIECZOREK *et al.*, 2005, ZENG *et al.*, 2004). Como esses sistemas são geralmente heterogêneos e complexos, otimizadores centralizados demandam uma maior replicação de informações globais. Esses aspectos não têm sido devidamente tratados na literatura, pois poucos trabalhos abordam o problema da otimização descentralizada. Em geral, é considerada apenas a descentralização do controle de execução (*i.e.*, envolvendo a possibilidade de delegação). Ainda assim, tal descentralização costuma ser restrita a um conjunto fixo de sítios especializados, em arquiteturas hierárquicas.

Na materialização de documentos AXML, a descentralização é um aspecto crítico e fundamental. A seguir, são apresentadas as características gerais de otimizadores que apresentam algum recurso para apoio à descentralização na execução de *workflows* distribuídos. É feita uma análise dos mecanismos oferecidos pelos principais sistemas da literatura, com ênfase no tipo de arquitetura utilizada para

controle tanto de execução como de otimização. Conforme será visto, esses trabalhos apresentam várias limitações, especialmente por serem baseados somente na orquestração de tarefas, limitando as possibilidades de colaboração entre os sítios.

3.4 OTIMIZADORES DESCENTRALIZADOS

Apesar de sua importância, a otimização da execução de *workflows* distribuídos ainda representa vários desafios para a pesquisa, especialmente em ambientes heterogêneos e dinâmicos. Alguns problemas merecem destaque, especialmente os resultantes da necessidade de colaboração entre sítios. Conforme visto na seção 3.3.2, otimizadores distribuídos podem estar organizados em uma estrutura hierárquica ou descentralizada. Em redes hierárquicas, existe uma estrutura fixa com um conjunto pré-definido de sítios que podem participar da execução de um *workflow*. Além disso, geralmente apenas o controle da execução de tarefas é descentralizado e não há possibilidade de colaboração para descoberta de informações, como, por exemplo, para a localização de serviços Web.

Na descentralização de um *workflow*, um ponto crucial consiste em técnicas para dividir adequadamente o *workflow* em partes menores, que possam ser tratadas individualmente. Um dos pioneiros em distribuição de *workflows*, o sistema Exotica/FMQM (de "*FlowMark on Message Queue Manager*") (MOHAN *et al.*, 1995, ALONSO *et al.*, 1995b) explora a idéia de quebrar o *workflow* em partes independentes que podem ser executadas por sítios autônomos. A comunicação entre sítios ocorre por mensagens assíncronas persistentes, usadas para disparar os próximos passos durante a execução. Para isso, o Exotica/FMQM utiliza o gerente de mensagens MQSeries, atualmente chamado de IBM WebSphere MQ (WEBSPHERE, 2007). A geração de planos de avaliação no Exotica/FMQM é estática e a divisão de um plano para distribuição de tarefas é baseada em perfis de usuários. Ou seja, os recursos são selecionados para grupos de tarefas relacionadas a um determinado usuário. Essas características limitam bastante a aplicação das técnicas de otimização do Exotica/FMQM em sistemas P2P.

A descentralização é interessante em vários aspectos, especialmente para tratar problemas complexos em ambientes dinâmicos. Por exemplo, o modelo de execução

descentralizada do Exotica/FMQM foi utilizado por ALONSO *et al.* (1995a) para apoiar a implementação de clientes com desconexão, em sistemas de computação móvel. Neste caso, a otimização é orientada a listas de tarefas prontas para execução e é apoiada por um conjunto de servidores. As atividades de um *workflow* são divididas entre clientes, que têm autonomia para executá-las sem a supervisão de um servidor. Essa autonomia permite que os clientes continuem a operar mesmo em períodos de disconexão com o sistema. Entretanto, a divisão das tarefas entre os clientes é realizada pelos usuários.

Como no Exotica/FMQM, geralmente otimizadores para *workflows* descentralizados contemplam apenas a distribuição do controle de execução. Além disso, a otimização costuma ser ou estática ou "*just-in-time*", ambas abordagens pouco adequadas para tratar o dinamismo, a larga escala e a complexidade dos sistemas P2P. A seguir, são descritas as principais abordagens utilizadas para tratar esses aspectos.

3.4.1 MENTOR

O sistema MENTOR (de "*Middleware for ENTerprise-Wide WORkflow Management*") (MUTH *et al.*, 1998, WODTKE *et al.*, 1996) aborda o problema de como dividir corretamente uma especificação de *workflow* para permitir execuções distribuídas. Para representar *workflows*, são adotados dois formalismos complementares: diagramas de atividades, nos quais são indicados fluxos de dados; e diagramas de estados, para expressar fluxos de controle definidos por regras do tipo Evento-Condição-Ação. Em um diagrama de estados, o usuário pode ainda especificar estados aninhados (*i.e.*, quando um estado representa uma composição de estados) e componentes ditos ortogonais, que podem ser executados em paralelo. Cada especificação de *workflow* engloba um diagrama de atividades e um diagrama de estados.

O cerne do MENTOR é um algoritmo que fragmenta especificações de *workflows* por meio de dois passos principais. No primeiro passo, o algoritmo pressupõe que já existe um mapeamento de atividades em sítios que irão executá-las. Assim, as partições são obtidas agrupando atividades atribuídas a um mesmo sítio. Vale ressaltar que, além de ignorar os problemas relativos à seleção de recursos, essa estratégia de divisão do *workflow* não trata problemas de desbalanceamento de carga.

Já no segundo passo de fragmentação de uma especificação de *workflow*, o sistema MENTOR usa uma análise mais sofisticada. Inicialmente, no diagrama de estados, ele identifica o sítio que executará cada atividade. Em seguida, o diagrama de estados é "ortogonalizado". Isto é, cada estado é substituído por um componente ortogonal, que consiste em um par de estados representando a entrada e a saída do estado original. São geradas transições que mantêm as ligações das transições entre os estados originais. Por fim, o MENTOR agrupa os componentes ortogonais em partições conforme o sítio executor, baseado na fragmentação do diagrama de atividades. A principal preocupação deste algoritmo é preservar no diagrama distribuído as propriedades de acionamento dos estados originais.

Como no Exotica/FMQM, a arquitetura do sistema MENTOR é baseada em um mecanismo de gerência de mensagens persistentes. Em particular, esse mecanismo é implementado pelo monitor de transações BEA *Tuxedo* (TUXEDO, 2001), que também garante o sincronismo das mensagens entre sítios durante execuções distribuídas. Informações sobre o andamento das instâncias de *workflows* são replicadas entre sítios. Em um cenário básico de distribuição, o MENTOR considera que o sincronismo entre sítios é obtido por inundação de mensagens. Como essa solução apresenta um alto custo de comunicação, é proposta uma estratégia que limita o envio de mensagens aos sítios interessados.

O sistema MENTOR visa apoiar sistemas de *workflows* em larga escala, com um grande número de instâncias concorrentes, impondo cargas pesadas de processamento no sistema de gerência de *workflows*. Nestes casos, são requisitos fortes a alta disponibilidade e a tolerância a falhas. Ao dividir um *workflow* para execução distribuída, o MENTOR tenta aproveitar o paralelismo de tarefas independentes entre si. Como no XCraft, o MENTOR explora o formalismo para garantir a correção da fragmentação de uma instância de *workflow*. Porém, o MENTOR não esclarece como são selecionados recursos quando vários sítios podem executar uma mesma atividade, nem trata da geração de alternativas de avaliação.

3.4.2 ADEPT-WFMS

O sistema ADEPT-WfMS (de "*Application Developement based on Encapsulation pre-modeled Process Templates – Workflow Management System*")

(BAUER e DADAM, 1997) visa apoiar a descentralização no controle da execução de *workflows* distribuídos. Em particular, o ADEPT-WfMS enfoca a redução dos custos de transferência de dados. A arquitetura geral desse sistema considera que servidores de *workflows* são acionados por clientes espalhados em diferentes sub-redes. Pressupõe-se que cada servidor representa uma sub-rede. Além disso, cada *workflow* é composto por um conjunto de passos a serem disparados por clientes.

A idéia fundamental do ADEPT-WfMS é permitir que o controle da execução de um *workflow* migre entre servidores para explorar a proximidade com os clientes. Para tanto, o ADEPT-WfMS divide o plano de avaliação de um *workflow* em partições, acrescentando operadores que indicam a migração de controle. Cada partição é mapeada em um servidor. Em uma abordagem bem interessante, o ADEPT-WfMS adota um algoritmo de fragmentação vertical de bases de dados relacionais para agrupar clientes em sub-redes. É construída uma matriz de afinidade a partir da probabilidade dos clientes executarem cada passo do *workflow*. Contudo, essa análise pressupõe que as sub-redes podem ser montadas sob demanda, o que raramente acontece na prática.

No ADEPT-WfMS, a geração de planos é baseada em um processo interativo e incremental. O mapeamento de partições em servidores é apoiado por um algoritmo guloso. Entretanto, a otimização é centralizada, estática e ignora tanto o agendamento da execução das tarefas como a carga de processamento dos clientes e servidores. Uma desvantagem importante do ADEPT-WfMS é que o usuário deve tomar muitas decisões sobre o mapeamento do *workflow* em clientes e servidores. Outro ponto negativo é a falta de detalhes sobre o cálculo dos parâmetros de custo. Na análise de desempenho das alternativas de avaliação, apenas as transferências de dados são consideradas. Isto é, a solução não contempla tempos de execução das tarefas.

Para superar as limitações do mapeamento estático de partições em servidores, BAUER e DADAM (2000) estenderam os conceitos do ADEPT-WfMS no sistema $ADEPT_{distribution}$. Neste caso, são gerados planos de execução com referências abstratas, vinculadas a fatores como a sub-rede de um determinado cliente ou o servidor selecionado para controlar outros passos do *workflow*. Essas referências são determinadas pelo usuário e traduzidas adequadamente em tempo de execução, conforme uma análise de custos. Contudo, vale destacar que no $ADEPT_{distribution}$ persistem os problemas encontrados na estratégia de otimização do ADEPT-WfMS.

3.4.3 GRIDFLOW

No contexto de *grids*, o sistema GridFlow (CAO *et al.*, 2003) oferece mecanismos para a especificação, execução e monitoração de *workflows* distribuídos. Por intermédio de uma interface gráfica, o usuário pode especificar *workflows* com diferentes padrões de controle de fluxo, descrever as tarefas a serem executadas e informar características dos serviços requisitados. Essas especificações são armazenadas em documentos XML. O GridFlow também é capaz de descobrir informações sobre serviços e sítios disponíveis em um *grid*, usando-as para simular o desempenho de uma instância de *workflow* antes de sua execução. A principal função do resultado de uma simulação é determinar um plano de avaliação a ser passado para a máquina de execução de *workflows*.

Ao realizar simulações, o GridFlow utiliza o ambiente PACE (de "*Performance Analysis and Characterization Environment*") (NUDD *et al.*, 2000) para estimar os custos dos recursos utilizados na execução de um *workflow*. O PACE considera quatro categorias de objetos envolvidos na análise de desempenho: aplicações, que dão a visão geral do sistema a ser modelado; subtarefas, que são partes seqüenciais independentes de uma aplicação, a serem executadas em paralelo quando possível; modelos de execução paralela, que descrevem o padrão de execução e comunicação dos passos de uma subtarefa; e recursos de *hardware*, que são os objetos elementares de um sistema, aos quais são atribuídos os passos de um modelo de execução paralela. Vale ressaltar que o PACE é voltado para a análise de desempenho de aplicações paralelas implementadas com MPI e PVM. Além disso, cabe ao usuário detalhar o modelo de desempenho de cada programa de uma simulação, pela linguagem $CHIP^3S$.

O GridFlow utiliza uma arquitetura hierárquica para otimizar a execução de *workflows*. Essa arquitetura é baseada no ARMS (CAO *et al.*, 2002), um sistema de agentes para controle da execução de tarefas distribuídas em *grids*. No ARMS, *grids* locais são agrupados para formar um *grid* global. Assim, sítios podem ser controladores locais ou globais. Uma especificação de *workflow* é feita com a perspectiva global, sendo composta por sub-*workflows* a serem executados em *grids* locais. Assim, a otimização no GridFlow se dá em duas fases:

(i) na fase global são determinados os controladores locais a serem acionados para a execução de cada sub-*workflow*, bem como a ordem de submissão aos controladores locais selecionados; e

(ii) na fase local, os controladores locais utilizam o otimizador Titan (SPOONER *et al.*, 2002) para agendar a execução das tarefas de cada sub-*workflow*. No Titan, a otimização é estática e baseada em algoritmos genéticos.

Observe que cabe ao usuário determinar como a execução de um *workflow* está dividida em sub-*workflows*. Além disso, a escolha dos controladores locais não considera transferências de dados entre sub-*workflows*, nem os custos envolvidos na delegação de tarefas. O otimizador Titan usa o PACE para adquirir informações sobre custos e estatísticas das tarefas de um sub-*workflow*.

Um aspecto interessante do GridFlow consiste no uso de funções de custo "*fuzzy*" para estimar o tempo de conclusão das tarefas levando em conta a carga dos sítios envolvidos (CAO *et al.*, 2003). Essas funções são baseadas em operações aplicadas a distribuições de probabilidades. Todavia, não é dito exatamente como os coeficientes de probabilidade são obtidos para cada tarefa de um *workflow*, nem é avaliado o impacto do tempo gasto no cálculo desses coeficientes na otimização.

O GridFlow permite ainda que um plano seja re-otimizado caso sejam verificados atrasos significativos nas previsões iniciais. Para isso, o plano é reenviado ao ambiente de simulação.

3.4.4 TRIANA

O sistema Triana (TRIANA, 2003, TAYLOR *et al.*, 2003, MAJITHIA *et al.*, 2004) oferece um ambiente gráfico para especificar *workflows* com execução distribuída, baseada em um conjunto de ferramentas para execução em *grids*. Entre essas ferramentas, destaca-se o sistema GAT, de "*Grid Application Toolkit*" (ALLEN *et al.*, 2005), que permite descrever especificações abstratas e dispõe de uma infra-estrutura básica de *grid* para agendar de tarefas. O Triana também utiliza a plataforma JXTA (JXTA, 2006, GONG, 2001) como infra-estrutura básica de comunicação P2P e

dispõe de mecanismos para descobrir serviços Web, por exemplo, a partir de repositórios UDDI.

No Triana, a execução de um *workflow* pode ser distribuída entre diferentes sítios remotos. Neste caso, o GAT pode ser acionado para selecionar automaticamente os recursos necessários para a execução das tarefas. Tal seleção é local e "*just-in-time*", a partir de um conjunto de critérios pré-definidos pelo usuário para cada tarefa. Cada sítio é representado por um serviço de controle chamado TCS (de "*Triana Controlling Service*"), que é ligado a um mecanismo de execução de *workflow*. A distribuição da execução é baseada em grupos de tarefas e considera que cada tarefa de um grupo é designada a um sítio distinto. O Triana implementa ainda um mecanismo P2P que permite transferir resultados intermediários diretamente entre sítios que colaboram na execução de um *workflow*. Todavia, tanto a fragmentação do *workflow* (em grupos de tarefas) como a seleção de sítios disponíveis para colaboração devem ser determinadas manualmente pelo usuário. Essa tarefa não é trivial, pois *grids* e sistemas P2P costumam ser arranjos complexos de sítios.

Vale destacar que a otimização no Triana não considera os custos e benefícios da delegação de tarefas. Como são tomadas decisões locais, geralmente não é possível identificar reduções nos custos de transferência de dados e não são analisadas as dependências entre tarefas para determinar prioridades de execução. Além disso, sítios que recebem delegações não podem envolver outros sítios na colaboração e não há tratamento para alterações dinâmicas na especificação de um *workflow*.

3.4.5 D2WMS

O sistema D2WMS (DIAS *et al.*, 2003) foi desenvolvido na Pontifícia Universidade Católica do Rio de Janeiro (PUC-Rio) e utiliza uma arquitetura hierárquica para apoiar execuções distribuídas de *workflows*. O D2WMS permite que instâncias de *workflows* sejam avaliadas incrementalmente por execuções parciais. O cenário-alvo são os ambientes com desconexão de rede, tais como as redes móveis. Esse sistema é baseado em um mecanismo de gerência de eventos que controla as interações entre sítios e usa uma estratégia de replicação otimista de dados para propagar informações sobre o andamento de instâncias de *workflows*.

Especificações de *workflows* são representadas internamente no D2WMS por um formalismo baseado em redes de Petri. Contudo, o D2WMS não descreve como *workflow* é dividido para distribuição, nem detalha os critérios de seleção de sítios. O modelo de execução considera que a comunicação entre sítios ocorre por meio de mensagens e notificação de eventos. O sistema também oferece recursos para descoberta de serviços. Prazos de expiração das tarefas são usados para identificar problemas decorrentes de longas desconexões. Embora o modelo de execução seja descentralizado e contemple explicitamente delegações de tarefas, aparentemente a otimização é centralizada.

O D2WMS permite que várias instâncias de uma mesma tarefa sejas disparadas em paralelo. A primeira instância a ser concluída provoca o cancelamento das demais. Por simplificação, delegações recursivas não são permitidas para essas múltiplas instâncias, visando evitar um grande volume de cancelamentos trafegados na rede. Também é considerada a compensação de tarefas, em caso de falhas. O D2WMS representa uma iniciativa importante na implementação de um modelo descentralizado para a execução de *workflows* distribuídos. O sistema D2WMS foi desenvolvido na linguagem Java e testado por roteiros Python (PYTHON, 2007).

Um outro trabalho conduzido pelo grupo do D2WMS é apresentado por VIEIRA e CASANOVA (2007), relativo à flexibilização da execução de *workflows* pelo tratamento de exceções. Os autores consideram que *workflows* são especificados por uma modelagem abstrata, com concretização postergada. O modelo de execução permite o uso de recursos alternativos na ocorrência de falhas e a delegação de partes de um *workflow*. É apresentada uma descrição detalhada das mensagens necessárias para garantir a continuidade de execuções distribuídas de *workflows* face à ocorrência de exceções. Porém, a escolha de recursos é reativa, sendo disparada essencialmente pelas exceções. Além disso, apenas o controle da execução pode ser delegado e a seleção dos coordenadores é aleatória. Também não são tratados problemas relativos à complexidade da geração de alternativas de avaliação.

3.4.6 GRAND

O sistema GRAND (acrônimo de "*Grid Robust Application Deployment*") (VARGAS *et al.*, 2004, 2005) consiste em uma plataforma baseada em redes

hierárquicas para a gerência da execução de *workflows* em *grids*. O sistema visa tratar aplicações que envolvam um grande número de tarefas, possivelmente com dependências de execução. A proposta consiste em uma arquitetura que permite distribuir a carga de submissão de tarefas e monitorar as respectivas execuções por meio de uma estrutura hierárquica de sítios. Também é apresentado um algoritmo para agrupar tarefas baseado em dependências, visando distribuir tarefas em sítios tal que seja explorada a proximidade dos dados utilizados.

O principal objetivo do sistema GRAND é evitar gargalos quando um grande número de tarefas é submetido ao *grid*. Nessa situação, vários problemas de escalabilidade surgem, principalmente em relação ao encaminhamento das tarefas para os sítios e o respectivo controle de execução.

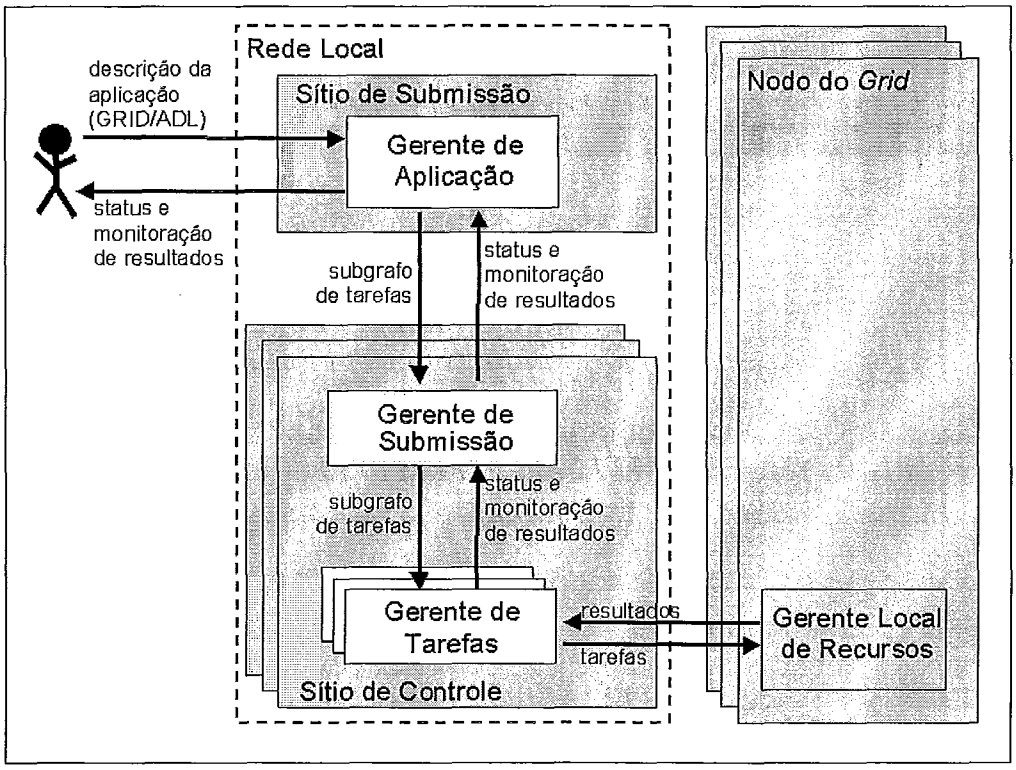


Figura 10. Arquitetura do sistema GRAND (VARGAS *et al.*, 2005).

A arquitetura do sistema GRAND é mostrada na Figura 10. Para descrever as tarefas de uma aplicação, utiliza-se a linguagem GRID-ADL (de "*Grid Application Description Language*"), que é baseada nas linguagens dos tradicionais sistemas Chimera (FOSTER *et al.*; 2002) e Condor DAGMan (THAIN *et al.*, 2005) para *grids*. Especificações em GRID-ADL são tratadas por um Gerente de Aplicação, que infere os

respectivos grafos de dependências entre as tarefas do *workflow*. Além disso, o Gerente de Aplicação divide cada grafo de dependências em grupos de tarefas (*i.e.*, subgrafos) a serem enviados para Gerentes de Submissão localizados em diferentes sítios. Por sua vez, cada Gerente de Submissão cria instâncias de Gerentes de Tarefas para controlar a execução das tarefas dos respectivos subgrafos. Considera-se ainda que cada sítio do *grid* possui um Gerente Local de Recursos, capaz de receber solicitações de um Gerente de Execução e de agendar tarefas a serem executadas localmente. Periodicamente, os Gerentes de Submissão se comunicam com o Gerente de Aplicação para relatar o andamento da execução. O agendamento das execuções de cada subgrafo é feito pelos respectivos Gerentes de Tarefas por meio de um algoritmo dinâmico e local.

As dependências entre tarefas são representadas por arquivos de dados. Ou seja, resultados intermediários são sempre armazenados. No GRAND, o Gerente de Aplicação tenta explorar essas dependências para agrupar tarefas, visando reduzir custos de comunicação. Cada grupo de tarefas é atribuído a um Gerente de Submissão.

Em avaliações empíricas, o sistema GRAND mostrou ganhos de desempenho importantes quando grandes volumes de tarefas são submetidos ao *grid*. Todavia, a escolha dos Gerentes de Submissão na execução de um *workflow* é aleatória. Além disso, as dependências consideradas são simples e não é previsto tratamento para mudanças na especificação de um *workflow*. Embora a divisão do *workflow* em subgrafos seja baseada nas dependências de dados entre tarefas, a geração de alternativas de avaliação no GRAND não leva em conta os custos da delegação.

3.4.7 *SELF-SERV*

Visando a execução de composições de serviços Web em cenários P2P, a plataforma *Self-Serv* (BENATALLAH *et al.*, 2005) implementa mecanismos para favorecer a adaptação da execução de *workflows* em sistemas de larga escala e ambientes dinâmicos. Nessa plataforma, serviços Web são catalogados para agilizar tanto a especificação de composições como a seleção de recursos. Serviços Web podem ser elementares ou compostos. Além disso, eles são agrupados em comunidades de serviços, que consistem essencialmente em classificações semânticas que permitem descrever especificações de alto nível a partir de referências abstratas. Essas referências são traduzidas em tempo de execução.

No *Self-Serv*, serviços compostos correspondem a *workflows* e são representados por diagramas de estados. Os serviços componentes de uma especificação de *workflow* podem ser tanto elementares como compostos. A passagem de parâmetros entre chamadas de serviços componentes pode ser especificada por expressões XPath, análogo ao adotado em documentos AXML. Entretanto, a seleção dos sítios para a execução de cada serviço componente é "*just-in-time*" e considera decisões locais.

O *Self-Serv* usa um modelo hierárquico de execução distribuída, o qual é baseado em orquestração de serviços Web. Um coordenador inicial dispara a execução distribuída, convocando coordenadores de segundo nível para controlar a execução de cada serviço componente. Fluxos de dados entre serviços são descritos em uma tabela de roteamento, que é utilizada para guiar mensagens de controle entre coordenadores de execução. Vale destacar que tal estratégia não considera os custos das transferências de dados entre execuções de serviços componentes, nem permite tirar proveito do agrupamento de execuções relacionadas para reduzir esses custos. Também são ignorados os custos das delegações de tarefas. Isto é, a escolha dos coordenadores é aleatória.

Pré e pós-condições de serviços são usadas para determinar os fluxos de controle entre coordenadores durante uma execução distribuída. Esses coordenadores notificam o coordenador inicial sobre o andamento das instâncias de serviços componentes. Por sua vez, o coordenador inicial constrói o resultado final e avisa o usuário sobre o fim da execução. A otimização é centralizada no coordenador inicial e pressupõe-se que não ocorrem modificações na especificação inicial.

3.4.8 ASKALON

O sistema ASKALON (DUAN *et al.*, 2005, WIECZOREK *et al.*, 2005) consiste em um amplo projeto para apoiar a execução distribuída e descentralizada de *workflows*. Um diferencial importante do ASKALON está na disponibilidade de várias ferramentas para especificar, executar e monitorar instâncias de *workflow*. Especificações de alto nível são descritas com a linguagem AGWL (de "*Abstract Grid Workflow Language*"), contemplando diversos padrões de controle de fluxo. Para serem executadas, essas especificações são convertidas automaticamente na linguagem CGWL (de "*Concrete Grid Workflow Language*"). O usuário pode também definir para cada tarefa um

conjunto de restrições a serem utilizadas para guiar o mapeamento de tarefas em recursos do *grid*. Todavia, nessa conversão, pressupõe-se que cada tarefa de uma instância de *workflow* é executada em um sítio distinto (DUAN *et al.*, 2005).

O ASKALON adota uma estratégia híbrida de otimização. Em um primeiro momento, é feita a seleção estática de recursos do *grid* para o *workflow* inteiro, por um algoritmo genético. Isso resulta na geração de um especificação concreta. Em seguida, um algoritmo heurístico "*just-in-time*" realiza o agendamento da execução das tarefas do *workflow* a partir de uma lista de tarefas prontas. Vale destacar que o ASKALON ignora possíveis agrupamentos de tarefas para determinar a delegação de tarefas, visando reduzir custos com transferências de dados. É utilizada uma arquitetura baseada em um controlador mestre que monitora e coordena outros controladores escravos, que geralmente não é adequada para sistemas P2P.

3.4.9 COMENTÁRIOS GERAIS

Existem outros sistemas que exploram técnicas de descentralização para a execução de *workflows* distribuídos. Por exemplo, o sistema Gridbus (YU e BUYYA, 2004) propõe um método hierárquico para o agendamento da execução de tarefas, cuja seleção de recursos é "*just-in-time*". Um componente de gerência de execução é atribuído a cada tarefa do *workflow*, sendo controlado por um coordenador global. Muitos desses sistemas são desenvolvidos para *grids*. Neste contexto, um aspecto crítico da otimização diz respeito às transferências de dados entre tarefas de uma instância de *workflow*. Algumas iniciativas implementam um mecanismo de persistência P2P, com localização automática de dados. É o caso do sistema DIET (ANTONIU *et al.*, 2005), que usa a plataforma JXTA (JXTA, 2006, GONG, 2001) como infra-estrutura básica para um ambiente de compartilhamento de bases de dados entre sítios. Esse ambiente é chamado JUXMEM. O sistema DIET é baseado em uma arquitetura hierárquica e visa especialmente a manutenção da consistência de dados compartilhados entre sítios.

Conforme discutido nos sistemas apresentados anteriormente, o apoio à descentralização na execução de *workflows* distribuídos ainda é incipiente. Muitos dos problemas enfrentados se devem à ausência de mecanismos que permitam ao otimizador analisar objetivamente o impacto da delegação, seja da execução ou mesmo da própria otimização. O otimizador deve ser capaz de tratar criteriosamente a complexidade da

busca por alternativas eficientes de execução distribuída. Esses aspectos não são satisfatoriamente implementados nos sistemas atuais. Muitos sistemas comerciais investiram em soluções relacionadas a serviços Web e *workflows*, como é o caso da Oracle e da IBM. Em particular, a plataforma *WebSphere MQ Workflow* (WEBSPHERE MQ WORKFLOW, 2005) engloba o gerente de mensagens assíncronas usado pelo sistema Exotica/FMQM. Esses sistemas costumam oferecer ferramentas para encapsular aplicações como serviços Web e descrever composições desses serviços com a linguagem WS-BPEL. Entretanto, soluções comerciais sofrem das mesmas limitações encontradas na pesquisa de *workflows* em *grids* e sistemas P2P.

Outro ponto crucial para a descentralização da execução de *workflows* distribuídos é a fragmentação adequada das especificações abstratas, visando quebrar o problema em partes menores e incentivar execuções paralelas. Otimizadores para *grids* amiúde trabalham com listas de tarefas prontas. Ou seja, os *workflows* são fragmentados em níveis, tal que cada nível possui tarefas independentes entre si. Essa abordagem costuma tornar o otimizador incapaz de detectar antecipadamente possibilidades de redução de custos de transferências de dados.

Uma inovação importante do XCraft consiste na proposta de uma álgebra que inclui operadores relativos à geração de planos de materialização. Esses operadores tratam basicamente da localização de serviços concretos, da delegação de tarefas e da delegação da otimização propriamente dita. Vale ressaltar ainda que esses operadores correspondem a serviços Web básicos para colaboração entre sítios. A álgebra do XCraft também distingue operadores abstratos que permitem ao otimizador avaliar incrementalmente os planos de materialização.

3.5 Análise de Desempenho de Serviços Web

O desempenho da execução de um *workflow* pode ser avaliado por diferentes métodos. Em geral, utiliza-se o tempo de resposta da execução para comparar planos alternativos, o qual costuma ser crítico para aplicações Web (CHERKASOVA *et al.*, 2003). Essa métrica também pode ser combinada com outros critérios, sejam objetivos (e.g., o preço do acesso aos recursos necessários) ou qualitativos, tais como confiabilidade e vulnerabilidade. Por exemplo, LIU *et al.* (2004a) propõem uma técnica

para combinar diferentes métricas de QoS na seleção de serviços Web equivalentes. São considerados parâmetros de alto nível e a idéia é normalizar métricas de QoS por intermédio de operações com matrizes.

Métodos analíticos tradicionais para estimar o desempenho de sistemas computacionais são baseados em um amplo detalhamento de fatores de custo, como uso de CPU e operações de entrada/saída de dados (RUBERG, 2001). Essa tendência é observada na análise de desempenho de *workflows* de alguns sistemas, como acontece na ferramenta PACE (NUDD *et al.*, 2000) do sistema ASKALON. Contudo, em se tratando de serviços Web e sistemas P2P, é impraticável coletar e manter atualizados parâmetros de custo e estatísticas tão detalhados. Nesses cenários, muitas vezes o otimizador tem que lidar com "caixas-pretas". Ou seja, ele deve se orientar por características observáveis dos serviços Web e por alguns parâmetros gerais de desempenho dos sítios. Além disso, o uso de serviços Web implica em operações específicas, tais como empacotamento de mensagens SOAP e validação de dados XML. Essas operações costumam ter impacto substancial no tempo de resposta de um serviço Web, conforme observado experimentalmente em RUBERG *et al.* (2004a). Todavia, esses fatores são ignorados pela maioria dos otimizadores para *workflows* baseados em serviços Web (AZEVEDO, 2003, BENATALLAH *et al.*, 2005, YU *et al.*, 2005). De fato, com o uso crescente de serviços Web, técnicas de otimização específicas dessa tecnologia têm se tornado cada vez mais necessárias (OUZZANI e BOUGHETTAYA, 2004).

No XCraft, explora-se uma arquitetura orientada a serviços Web e documentos AXML para coletar estatísticas e parâmetros de custo. Cada sítio publica um conjunto de serviços básicos para colaboração, permitindo que outros sítios adquiram essas informações. O desempenho de planos de materialização é estimado pelo tempo de resposta, cujo cálculo considera as principais características tanto da invocação como da execução de serviços Web, conforme apresentado em RUBERG *et al.* (2004a).

Vale ressaltar que os critérios usados para selecionar provedores de serviços Web podem apresentar resultados bastante divergentes. MENDONÇA e SILVA (2005) apresentam uma avaliação experimental abrangendo cinco diferentes políticas de seleção de provedores de serviços Web. São elas: aleatória, por desempenho de HTTP-*ping*, com execução paralela, pelo melhor desempenho na última execução e pelo

melhor desempenho na mediana das últimas execuções. Na seleção com execução paralela, múltiplas instâncias do serviço Web são disparadas em diferentes sítios, sendo utilizado o sítio vencedor (*i.e.*, o mais rápido em prover o resultado). Observe que essa estratégia possui um alto custo total, pois tende a sobrecarregar os sítios que participam do sistema. Enquanto a estratégia aleatória se mostrou ineficiente em geral, a com execução paralela foi vantajosa somente quando os clientes possuem grande capacidade computacional. Em suma, MENDONÇA e SILVA (2005) mostram que é importante a escolha de critérios objetivos para a seleção de provedores de serviços Web.

Basicamente, uma análise de desempenho pode ser baseada em técnicas de predição ou em um modelo de custo. A predição de desempenho consiste em aplicar um tratamento matemático ao histórico de execuções anteriores dos componentes de *software* observados. A ferramenta *Prophesy* (JANG *et al.*, 2005) utiliza técnicas de mineração de dados para construir um modelo de predição de desempenho de recursos em *grids*. Em uma avaliação experimental, resultados obtidos com a *Prophesy* mostraram que aplicações com ênfase na transferência de dados são passíveis de ganhos significativos de desempenho com a otimização. Isso porque a seleção de recursos com boa conectividade (*i.e.*, ligados por enlaces rápidos) pode acelerar a transferência de dados. Muitos otimizadores selecionam recursos apenas considerando a carga dos sítios, a partir de uma lista de tarefas prontas. Contudo, resultados experimentais de JANG *et al.* (2005) mostram que a seleção baseada em análise de desempenho é mais eficiente. Em WU *et al.* (2006), a ferramenta *Prophesy* é integrada a um ambiente gráfico que oferece recursos para o usuário descrever funções que representam o desempenho de programas.

Por outro lado, modelos analíticos de desempenho reúnem funções de custo que estimam o tempo (de resposta ou total) gasto na execução de tarefas. Na análise de *workflows* com serviços Web, também deve ser considerada a carga de processamento dos sítios. Ou seja, geralmente duas tarefas atribuídas a um mesmo sítio são executadas seqüencialmente. Além disso, o desempenho de um sítio é penalizado se ele é encarregado da execução de várias tarefas. Nesta tese, é apresentado um modelo de custo que abrange os principais padrões de fluxos de controle e de dados, bem como diferentes estratégias de agendamento da execução de tarefas.

3.6 CONSIDERAÇÕES FINAIS

Na materialização de documentos AXML, a otimização trata essencialmente da descentralização de tarefas, da seleção de sítios para execução de serviços Web e do agendamento da invocação de chamadas de serviços. Esses problemas abrangem ainda vários desafios simultâneos, como a distribuição de carga de processamento e a redução dos custos de transferência de dados. Conseqüentemente, a complexidade resultante é difícil mesmo para cenários bem simples. Portanto, torna-se obrigatório o uso de heurísticas para lidar com tal complexidade, especialmente em sistemas dinâmicos, nos quais o tempo gasto em otimização deve ser reduzido.

Todavia, heurísticas costumam ser efetivas apenas para aplicações ou configurações específicas. Assim, a principal questão na materialização de documentos AXML consiste em definir uma estratégia heurística genérica, que seja capaz de priorizar pontos relevantes para a otimização, como os fluxos de dados entre chamadas de serviços. Métodos puramente gulosos ou oportunistas tendem a ignorar indiscriminadamente esses aspectos. Para explorar estratégias efetivas de otimização, é essencial a adoção de uma representação concisa para as especificações de *workflows*. A seguir, é apresentado um formalismo canônico para expressar os padrões de fluxos de controle e dados encontrados em um documento AXML.

Capítulo 4

UM MODELO CANÔNICO PARA REPRESENTAR A MATERIALIZAÇÃO DE DOCUMENTOS AXML

Este capítulo apresenta um formalismo canônico para representar os fluxos de controle e de dados de um documento AXML. São especificadas regras para verificação de validade, eliminação de redundâncias e atualização dinâmica eficiente com respostas ativas. Além disso, a partir do formalismo proposto, são descritas as principais possibilidades de materialização de um documento AXML em um sistema P2P.

4.1 INTRODUÇÃO

Essencialmente, documentos AXML contêm elementos intencionais (*i.e.*, chamadas de serviços Web) que apontam para informações distribuídas em um sistema P2P. Portanto, para que o conteúdo de um documento AXML possa ser utilizado, ele precisa antes ser materializado pela invocação das chamadas de serviços do documento. Esse processo de materialização AXML é influenciado principalmente pelos relacionamentos entre elementos intencionais, que representam restrições (implícitas e explícitas) na ordem de invocação das chamadas de serviços. Vale mencionar que a linguagem AXML permite especificar restrições de invocação que correspondem a padrões de fluxos de controle e de dados encontrados em sistemas de *workflows* (AALST *et al.*, 2000).

Dependências de invocação são o tipo mais comum de restrição entre as chamadas de serviços de um documento AXML. Elas ocorrem quando uma chamada de serviço usa o resultado de outra chamada como parâmetro de entrada. Basicamente, essas dependências representam relacionamentos do tipo "produtor × consumidor". Ou seja, elas implicam fluxos de dados entre chamadas de serviços. Além desses relacionamentos, também podem haver dependências de serviços (RUBERG *et al.*, 2004a). Esse tipo de dependência ocorre quando alguma informação sobre o serviço Web apontado por uma chamada de serviço depende do resultado de uma outra chamada. Contudo, nesta tese, considera-se que a localização de serviços Web é postergada e tratada como um aspecto complementar à otimização, a ser atacado dinamicamente por intermédio da colaboração P2P. Assim, a representação das restrições de invocação das chamadas de serviços de um documento AXML não contempla dependências de serviços.

Além de dependências de dados, documentos AXML podem conter conseqüências de invocação decorrentes de chamadas colaterais. Neste caso, o usuário indica que a invocação de uma chamada de serviço deve disparar automaticamente outra chamada. Em geral, os diversos tipos de restrição de invocação de um documento AXML formam grafos complexos, devido principalmente à ocorrência de dependências compartilhadas e de chamadas colaterais. Assim, para controlar a materialização de documentos AXML, o formalismo tradicional baseado em árvores não é adequado para

expressar os relacionamentos entre chamadas de serviços. Neste capítulo, é apresentado um formalismo canônico para representar as restrições de invocação de um documento AXML em uma estrutura chamada de grafo de dependências. São propostos critérios para eliminar restrições redundantes e determinar a validade de um grafo de dependências em relação à ocorrência de bloqueios perpétuos e ciclos infinitos de execução. Além disso, são apresentadas técnicas eficientes para atualizar um grafo de dependências em tempo de execução, visando lidar com respostas ativas. Por fim, é apresentado um modelo P2P para a materialização de documentos AXML baseado no formalismo proposto.

4.2 DEPENDÊNCIAS DE INVOCAÇÃO

Em um documento AXML, é possível definir dependências entre chamadas de serviços, impondo restrições de precedência de invocação durante o processo de materialização. Basicamente, isso ocorre quando uma chamada de serviço precisa consumir o resultado de uma outra chamada como parâmetro de entrada. As chamadas de serviços Web que produzem tais resultados são ditas parâmetros intencionais.

As dependências de invocação decorrentes de parâmetros intencionais existem porque, ao se considerar um sistema P2P aberto, convém lembrar que serviços Web regulares geralmente desconhecem o formato de dados AXML. Ou seja, esses serviços não são capazes de processar corretamente entradas contendo parâmetros intencionais. Assim, esses parâmetros precisam ser materializados antes da chamada de serviço que os consumirá.

Por outro lado, em sistemas com sítios *ActiveXML*, alguns serviços Web podem ser capazes de lidar com dados AXML, dispensando a invocação prévia de parâmetros intencionais. Porém, vale destacar que outros fatores também podem influenciar a capacidade de um serviço Web de materializar parâmetros intencionais, tais como segurança, desempenho e preferência. Logo, no caso geral, cabe ao usuário especificar quando parâmetros intencionais devem ser materializados *a priori* ou não. Isso pode ser feito por indicação em um atributo do elemento "sc". Contudo, geralmente serviços Web regulares prevalecem em sistemas P2P. Portanto, por simplicidade, esta tese pressupõe que parâmetros intencionais são sempre materializados *a priori*, provocando

dependências de invocação. Esses parâmetros podem ser concretos ou não, conforme descrito a seguir.

4.2.1 PARÂMETROS INTENCIONAIS CONCRETOS

A forma mais simples de especificar uma dependência de invocação em um documento AXML é com o aninhamento de elementos intencionais nas sub-árvores que representam os parâmetros de entrada de uma chamada de serviço. Quando isso ocorre, diz-se que os elementos aninhados são parâmetros intencionais concretos (ou, simplesmente, parâmetros concretos). Por exemplo, na Figura 2, as chamadas de serviços sc2 e sc3 estão aninhadas na chamada sc1. Assim, na materialização do documento *SwapWorkspace*, para que o serviço *GetSwapStatus* (da chamada sc1) possa ser invocado, é preciso antes obter o resultado tanto da chamada para *GetCurrentSwaps* (*i.e.*, sc2) como da chamada para *GetSwapLimit* (*i.e.*, sc3). Ou seja, as chamadas sc2 e sc3 são parâmetros concretos de sc1. Este tipo de relacionamento entre chamadas de serviços de um documento AXML é definido a seguir.

DEFINIÇÃO 3: Seja um documento AXML D e um nodo de $v_i \in SC_D$. Um nodo v_j é dito um parâmetro intencional concreto de v_i se e somente se:

- $v_j \in SC_D$ e v_j é um nodo descendente de v_i em D; e
- não existe nenhum nodo $v_x \in SC_D$ tal que v_x seja ao mesmo tempo um descendente de v_i e um ancestral de v_j em D.

O termo $\nabla(v_i)$ representa o conjunto de todos os parâmetros concretos do nodo v_i, tal que $\nabla(v_i) \subseteq SC_D$.

Verificar se uma chamada de serviço é um parâmetro concreto de outra chamada é simples. Essa verificação pode ser feita antecipadamente, por uma análise estática, quando o documento é carregado e/ou atualizado em um sítio *ActiveXML*.

No exemplo da chamada de serviço sc1, o aninhamento dos parâmetros concretos representados pelas chamadas sc2 e sc3 corresponde ao padrão de sincronização de fluxos de controle em *workflows*. Parâmetros concretos também podem representar seqüências, como ocorre com sc9 e sc10 no documento *SwapWorkspace* da Figura 2.

4.2.2 PARÂMETROS INTENCIONAIS NÃO-CONCRETOS

Em um documento AXML, outra forma de determinar dependências de invocação consiste em usar consultas XPath na especificação dos parâmetros de entrada de uma chamada de serviço. Cada consulta XPath seleciona elementos que devem ser passados no parâmetro de entrada para a invocação da chamada de serviço. Entretanto, essas consultas se referem ao conteúdo materializado do documento AXML. Portanto, é preciso identificar quais chamadas de serviços podem contribuir para o conteúdo que é passado como parâmetro de entrada. O resultado dessas chamadas deve ser materializado para compor o parâmetro de entrada. Por exemplo, no documento *SwapWorkspace* (da Figura 2), o parâmetro de entrada foreign_currency da chamada de serviço sc7 é especificado como:

```
<param name="foreign_currency">
   <xpath>/current_contract/principal/currency</xpath>
</param>
```

No momento da invocação de sc7, a expressão de caminho "/current_contract/principal/currency" é avaliada no documento *SwapWorkspace* e os elementos selecionados são usados para compor o parâmetro foreign_currency. Se a seleção envolver algum conteúdo a ser gerado por elementos intencionais, os resultados de tais elementos são materializados antes da invocação de sc7. Neste exemplo, embora a chamada de serviço sc4 não esteja explicitamente aninhada em sc7, a invocação de sc7 depende de sc4, pois sc4 contribui para o parâmetro foreign_currency. Este caso indica um outro tipo de dependência de invocação, definido a seguir:

DEFINIÇÃO 4: Sejam D um documento AXML, v_i um nodo de SC_D e q uma expressão XPath dentro de um parâmetro de entrada de v_i. Um nodo v_j é dito um parâmetro intencional não-concreto de v_i se e somente se:

- $v_j \neq v_i$ e v_j consta no resultado de $\Phi(q)$, onde $\Phi(q)$ é uma função que retorna o conjunto de chamadas de serviços que contribuem para q em D, tal que $\Phi(q) \subseteq SC_D$; e
- não existe nenhum nodo v_x em SC_D tal que v_x seja um ancestral de v_j em D.

O termo $\vec{\nabla}(v_i)$ denota o conjunto de todos os parâmetros não-concretos de v_i, onde $\vec{\nabla}(v_i) \subseteq \Phi(q)$.

Vale destacar que uma chamada de serviço pode contribuir para uma expressão XPath sem necessariamente estar entre os elementos selecionados pela expressão, pois o critério de contribuição da função Φ se baseia no resultado da chamada de serviço. Isto é, se o resultado de uma chamada pode ser selecionado pela expressão XPath, então essa chamada é considerada contribuinte. A função Φ pode adotar diferentes abordagens. Por exemplo, ela pode considerar os tipos de dados dos parâmetros de saída dos serviços Web, conforme especificados nos respectivos arquivos WSDL. Na prática, a análise de Φ se baseia essencialmente na localização das chamadas de serviços no documento AXML, já que o resultado de uma invocação é sempre inserido no documento como um nodo irmão da chamada que o originou. Em todo caso, a expressão XPath de um parâmetro não-concreto é sempre avaliada antes da invocação da chamada de serviço, para que se obtenha as entradas necessárias.

Parâmetros intencionais não-concretos (ou, simplesmente, parâmetros não-concretos) constituem um recurso interessante na especificação de documentos AXML. Eles permitem, por exemplo, o compartilhamento do resultado de chamadas de serviços e podem representar os padrões de seqüência, divisão paralela e sincronização de fluxos de controle em *workflows*. Contudo, geralmente é necessária uma análise sofisticada para identificar o conjunto de chamadas que podem contribuir para uma expressão XPath. Essa análise é abordada em ABITEBOUL *et al.* (2004b).

Ao lidar com parâmetros não-concretos, alguns complicadores importantes devem ser tratados. Em especial, esses parâmetros podem causar ciclos de dependências, levando a um bloqueio perpétuo. Nesta teste, quando a ocorrência de ciclos é verificada, pressupõe-se que ou eles são quebrados antes do processo de otimização, ou o processo de materialização é abortado. Para detectar esse problema, na seção 4.4.3 são definidos critérios de validade para o grafo de dependências de um documento AXML.

4.2.3 TRANSITIVIDADE

As definições 3 e 4 podem ser generalizadas para representar o conjunto de todos os parâmetros intencionais de uma chamada de serviço, sejam eles concretos ou não. Tal conjunto indica o grau de dependência de uma chamada de serviço na materialização do documento AXML.

DEFINIÇÃO 5: Sejam os nodos v_i e v_j em um documento AXML D. Diz-se que v_j é um parâmetro intencional de v_i, indicado por $v_j \rightarrow v_i$, se e somente se $v_i \in \nabla(v_i) \cup \vec{\nabla}(v_i)$. Além disso, o termo *fanIn*(v_i) representa o número de parâmetros intencionais de v_i em D, que corresponde a $\left|\nabla(v_i) \cup \vec{\nabla}(v_i)\right|$. Analogamente, o termo *fanOut*(v_j) indica o o número de nodos de D dos quais v_j é um parâmetro intencional.

É importante observar que a Definição 3 (e, por conseqüência, a Definição 5) não considera parâmetros concretos por transitividade. Isto é, dadas as chamadas de serviços *a*, *b* e *c*, tal que *b* está aninhada em *a* e *c* está aninhada em *b*, então *c* não é um parâmetro intencional de *a*. A idéia desta simplificação decorre do fato de que dependências transitivas não existem na sua forma ativa após a primeira invocação, que é disparada pela dependência mais próxima. Neste caso, a dependência transitiva de *a* em relação a *c* é claramente redudante, pois *a* não poderia ser invocada antes de *b*, que também depende de *c*. Tal redundância também pode acontecer com parâmetros não-concretos, quando algumas chamadas de serviços em Φ são aninhadas em um documento AXML. Analogamente, para reduzir o número de redundâncias, a Definição 4 considera apenas as chamadas de serviços de primeiro nível. Por exemplo, supondo-se que os parâmetros text e input_format da chamada de serviço sc9 na Figura 2 sejam reescritos como a seguir:

```
<param name="text"><xpath>//</xpath></param>
<param name="input_format">XML</param>
```

Então, potencialmente todas as chamadas de serviços do documento *SwapWorkspace* podem contribuir para o resultado da expressão XPath em text (*i.e.*, considerando-se que ciclos são devidamente eliminados). Em particular, tanto sc1 como

sc2 estão em Φ(//). Porém, de acordo com a Definição 4, somente sc1 é um parâmetro não-concreto de sc9, já que sc2 está aninhada em sc1. Todavia, observe que continua existindo um relacionamento implícito entre sc2 e sc9, cuja representação é relevante para a otimização do processo de materialização.

DEFINIÇÃO 6: Sejam os nodos v_i e v_j em um documento AXML D. O nodo v_j é um parâmetro transitivo de v_i, indicado por $v_j \rightarrow^* v_i$, se e somente se e somente se existe um caminho da forma $v_j \rightarrow v_{x1} \rightarrow ... \rightarrow v_{xn} \rightarrow v_i$ em D, com $n \geq 1$. O comprimento deste caminho é dado por $n+1$ e representa o alcance da transitividade de v_j.

Ao se considerar todas os parâmetros intencionais de um documento AXML, podem aparecer redundâncias que não são eliminadas por definição. Conforme apresentado na seção 4.4.4, o conceito de transitividade é essencial para a eliminação dessas redundâncias.

Vale mencionar que o parâmetro intencional transitivo com o maior alcance determina o caminho crítico absoluto de um documento AXML. Este caminho consiste na maior seqüência de invocações do documento. Na materialização de um documento AXML, outro caminho de transitividade importante é o com o maior tempo de execução.

4.3 CHAMADAS COLATERAIS

Enquanto o aninhamento de elementos intencionais e os parâmetros definidos por consultas representam dependências de invocação, chamadas colaterais especificadas pelo atributo followed_by correspondem a gatilhos que devem ser disparados automaticamente na materialização de um documento AXML. Chamadas colaterais em um documento AXML implicam em conseqüências de invocação e provocam novas instâncias de chamadas de serviços durante a materialização.

DEFINIÇÃO 7: Um documento AXML com chamadas colaterais é denotado pela expressão $\langle D, \Rightarrow \rangle$, onde $\Rightarrow$ consiste em um mapeamento parcial da forma $\Rightarrow: SC_D \times SC_D$. Assim,

sejam os nodos v_i e v_j em D, com $v_i \neq v_j$, o termo $v_i \Rightarrow v_j$ indica que v_j é uma chamada colateral de v_i.

Por simplicidade, nesta tese pressupõe-se que uma chamada de serviço só pode apontar para apenas uma única chamada colateral. Contudo, chamadas colaterais também podem disparar outras chamadas do documento AXML, que por sua vez podem ter suas próprias chamadas colaterais e assim por diante. Logo, as conseqüências de invocação de um documento AXML podem ser transitivas, como definido a seguir.

DEFINIÇÃO 8: Sejam as chamadas de serviços v_i e v_j de um documento AXML $\langle D, \Rightarrow \rangle$. O nodo v_j é dito uma chamada colateral transitiva de v_i, denotado por $v_i \Rightarrow^* v_j$, se e somente se existe um caminho da forma $v_i \Rightarrow v_{x1} \Rightarrow ... \Rightarrow v_{xn} \Rightarrow v_j$ em D, com $n \geq 1$.

Vale destacar que chamadas colaterais denotam uma seqüência de invocações que não possuem necessariamente dependências de dados entre si. Por outro lado, dependências de invocação estão intrinsecamente relacionadas a fluxos de dados que alimentam parâmetros de entrada de chamadas de serviços. A distinção entre dependências e conseqüências entre processos também é feita em linguagens para *workflows*. São exemplos os construtores "*precede*" e "*enable*" da linguagem *ActivityFlow* (LIU *et al.*, 2004b), além dos "*enablingFlows*" do sistema Vortex (HULL *et al.*, 2000). Entre os padrões para especificação de *workflows*, chamadas colaterais correspondem à seqüência e à criação de múltiplas instâncias sem sincronização (pois cada chamada colateral dispara uma nova invocação do elemento intencional apontado).

4.4 GRAFO DE DEPENDÊNCIAS

Os diferentes relacionamentos entre as chamadas de serviços de um documento AXML formam teias de dependências e conseqüências de invocação. Nesta tese, esses relacionamentos são representados por um formalismo baseado em grafos orientados acíclicos (ou DAG, de "*directed acyclic graph*"). Vale mencionar que parâmetros não-concretos e chamadas colaterais resultam em restrições de invocação que podem ser arbitrariamente complexas. Tais relacionamentos não são naturalmente expressos em

uma estrutura de árvore. Assim, uma representação baseada em DAG é mais adequada tanto para representar as chamadas de serviços de um documento AXML como para a análise das restrições de invocação, especialmente para a verificação de ciclos e de transitividade de parâmetros não-concretos.

Todas as restrições de invocação entre as chamadas de serviços de um documento AXML são concisamente descritas em um grafo de dependências. Este grafo constitui a principal entrada da estratégia de otimização proposta nesta tese. Vale salientar que, apesar do nome, um grafo de dependências abrange tanto dependências (concretas ou não) como chamadas colaterais. Contudo, há ênfase nas dependências de invocação porque essas restrições refletem os parâmetros de entradas das chamadas de serviços, que são um aspecto essencial em documentos AXML. Por outro lado, chamadas colaterais são derivadas de anotações opcionais.

4.4.1 Estados de uma Chamada de Serviço Web

Os componentes básicos de um grafo de dependências são as chamadas de serviços Web do documento AXML, as quais são representadas por nodos do grafo. Essas chamadas constituem elementos dinâmicos que podem assumir diferentes estados ao longo do processo de materialização, conforme ilustrado no diagrama da Figura 11.

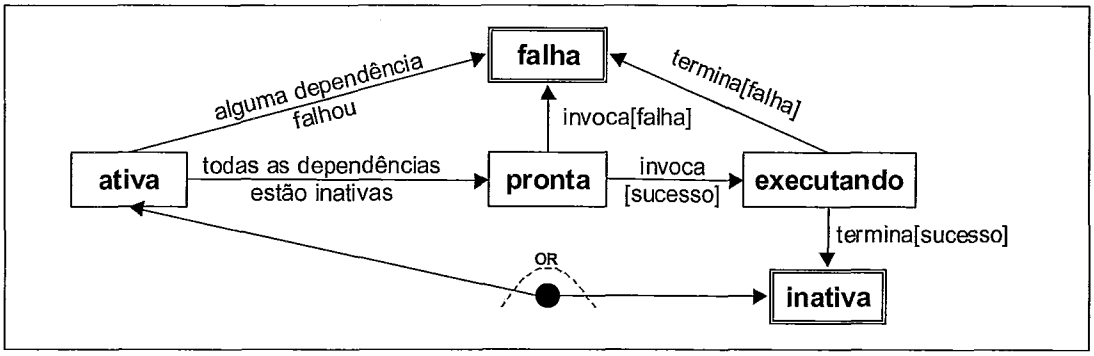


Figura 11. Diagrama de estados de chamadas de serviços Web.

Inicialmente, uma chamada de serviço pode estar ativa ou inativa. Chamadas ativas devem ser invocadas na materialização do documento AXML, enquanto chamadas inativas são tratadas como dados XML regulares. Uma chamada de serviço pode ser considerada inativa por diversas razões, tais como por ter uma especificação AXML incorreta, pela escolha do usuário ou porque alguma análise preliminar indica

que sua invocação é desnecessária para a materialização do documento AXML. Por exemplo, isso ocorre quando se deseja materializar apenas partes do documento, no processamento de consultas com predicados de seleção.

Uma chamada ativa se torna pronta quando todos os seus parâmetros intencionais estão inativos. Somente chamadas prontas podem ser invocadas, passando assim para o estado "executando". A execução de uma chamada de serviço dura até que o resultado de sua invocação seja devidamente inserido no documento AXML. Ao terminar sua execução, a chamada de serviço se torna inativa. Por outro lado, se alguma dependência da chamada de serviço falhar, bem como se houver problemas seja na sua invocação ou execução, então ela assume o estado "falha". Na Figura 11, os estados "inativa" e "falha" são representados por retângulos duplos para destacá-los como estados terminais (*i.e.*, quando as chamadas de serviços se tornam estáveis).

4.4.2 DEFINIÇÃO PRELIMINAR DO GRAFO

Basicamente, um grafo de dependência pode ser obtido por uma análise estática a partir da especificação do documento AXML, quando ela é armazenada ou alterada. Vale destacar que usuários podem especificar inadvertidamente documentos AXML contendo bloqueios perpétuos e ciclos infinitos de execução. Por isso, nesta tese é apresentada inicialmente uma definição geral para grafos de dependências. Em seguida, são propostas regras de validade para grafos de dependências, visando restringí-los a casos acíclicos. Assim, temos que:

DEFINIÇÃO 9: O grafo de dependências de um documento AXML $\langle D, \Rightarrow \rangle$ é denotado pela expressão $\Delta = \langle G, \otimes, \vec{E}, \varepsilon \rangle$, onde G é um grafo orientado formado por um conjunto V de nodos, tal que $V \subseteq SC_D$, e por um conjunto E de arestas. O termo $\otimes$ denota o subconjunto de nodos persistentes de Δ, tal que um nodo v_i pertence a $\otimes$ se e somente se $v_i \in \xi$ ou $fanOut(v_i) > 1$. As arestas de E podem ser simples ou colaterais, da seguinte forma:

- para quaisquer nodos v_i e v_j em V, existe uma aresta simples $v_i \rightarrow v_j$ em E se e somente se v_j é um parâmetro intencional de v_j em D; e

- o termo $\vec{E}$ representa o subconjunto de arestas colaterais de E, tal que para quaisquer nodos v_i e v_j em V, existe uma aresta colateral $v_i \Rightarrow v_j$ em $\vec{E}$ se e somente se $v_i \Rightarrow v_j$ ocorre em $\langle D, \Rightarrow \rangle$.

Por fim, a função ε associa cada nodo de V a um estado em $\{ativa, inativa, falha\}$.

É importante observar que um grafo de dependências determina uma ordem parcial na invocação das chamadas de serviços de um documento AXML, pois existem chamadas independentes entre si e que podem ser executadas em paralelo. Além disso, apenas os estados iniciais e terminais das chamadas de serviços são considerados na representação do grafo.

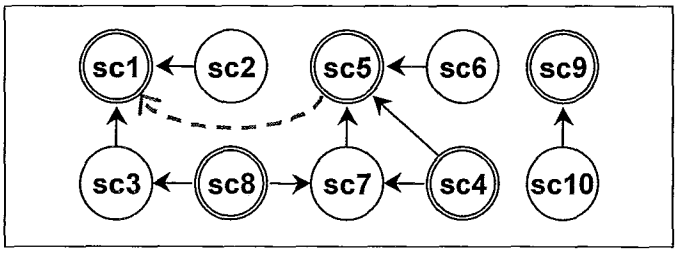


Figura 12. Grafo de dependências do documento *SwapWorkspace*.

A Figura 12 apresenta o grafo de dependências obtido do documento *SwapWorkspace* da Figura 2 (na seção 2.2.1). Os nodos do grafo são representados por círculos rotulados com as identificações das respectivas chamadas de serviços. Círculos com linhas duplas indicam nodos persistentes e flechas tracejadas representam arestas colaterais. Neste exemplo, pressupõe-se que todas as chamadas de serviços estão ativas (caso contrário, nodos inativos seriam indicados em cinza e nodos com falha por círculos cortados com um traço diagonal).

4.4.3 CONDIÇÕES DE VALIDADE DO GRAFO

Usuários podem inadvertidamente especificar documentos AXML contendo alguns ciclos maliciosos. Isto é, ciclos infinitos de execução formados por seqüências de chamadas colaterais e ciclos de dependências que causam bloqueios perpétuos. Nesta tese, grafos que contêm essas ocorrências são considerados inválidos, já que não podem ser materializados corretamente.

A Figura 13 ilustra os padrões básicos de ciclos maliciosos entre chamadas de serviços (representadas por scX, scY, scW e scZ) em um documento AXML. Basicamente, o subgrafo de arestas simples de um grafo válido deve ser acíclico, para evitar bloqueios perpétuos (Figura 13.a). Igualmente, o subgrafo de arestas colaterais não deve conter ciclos como os da Figura 13.c, garantindo que a materialização tenha um ponto de término.

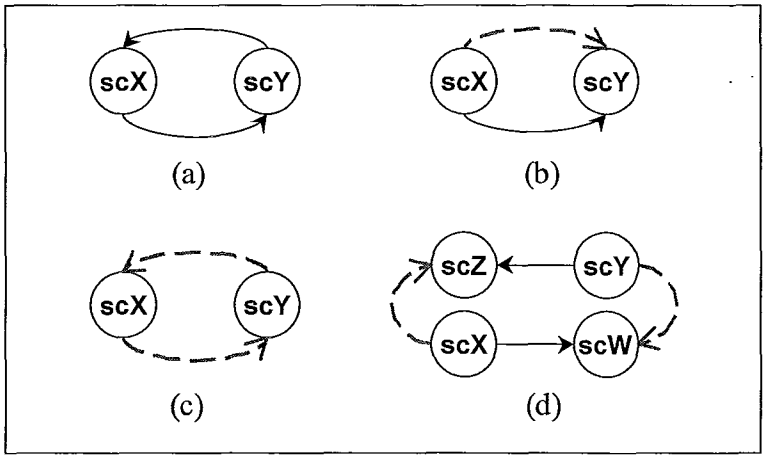


Figura 13. Padrões básicos de ciclos maliciosos em um documento AXML.

Embora a validade de um grafo de dependências esteja relacionada essencialmente à presença de ciclos, devem ser considerados apenas certos tipos de ciclos no grafo. Isso ocorre porque dependências de invocação e chamadas colaterais denotam tipos de arestas distintas entre nodos. Além disso, chamadas colaterais não implicam em dependências de dados. Assim, ciclos direcionados contendo tanto arestas simples como colaterais não geram bloqueios perpétuos ou problemas de terminação. Entretanto, essas arestas podem contribuir indiretamente para a formação de ciclos maliciosos. Em geral, um ciclo infinito de execução ocorre se um nodo é uma chamada colateral de alguma de suas dependências, como mostrado na Figura 13.b. Isso também ocorre caso uma seqüência alternada de arestas simples e colaterais formem um ciclo malicioso, como na Figura 13.d.

Observe que os padrões proibidos da Figura 13 também são aplicáveis a restrições transitivas. Isto é, se $v_i \rightarrow^* v_j$, então não se pode ter $v_j \rightarrow^* v_i$ no grafo. Resumindo, pode-se concluir que:

(i) considerando a direção das arestas simples e colaterais, elas não formam ciclos maliciosos entre si; e

(ii) se arestas simples e colaterais são combinadas com direções invertidas, elas podem contribuir para formar ciclos infinitos de execução.

O item *(ii)* indica que para verificar se a construção do grafo de dependências garante o término do processo de materialização, deve-se considerar arestas colaterais como arestas simples com a direção invertida. Com isso, seja o componente sequencial de um grafo de dependência obtido pelo grafo resultante da substituição de todas as arestas colaterais por arestas simples invertidas. Então, a verificação de validade de um grafo de dependências é dada pelo seguinte.

DEFINIÇÃO 10: Um grafo de dependências Δ é válido se e somente se:

- o subgrafo com todas as arestas simples de Δ é um digrafo acíclico; e
- o componente sequencial de Δ é um digrafo acíclico.

O primeiro item da Definição 10 evita bloqueios perpétuos e o segundo trata de ciclos infinitos de execução. É importante distinguir esses problemas porque o usuário pode precisar relaxar a noção de validade do grafo. Por exemplo, um caso comum de relaxamento consiste em permitir que a terminação da execução seja garantida por um ponto fixo pré-determinado (*i.e.*, que ciclos de execução sejam limitados a número determinado de vezes).

A validade de um grafo de dependências pode ser verificada quando o documento AXML é criado ou alterado. Essa verificação possui complexidade assintótica $O(|V|^3)$, baseada na complexidade de tempo para calcular o fechamento transitivo de um grafo pelo algoritmo de Warshall (AHO *et al.*, 1983). Vale mencionar que, quando há respostas ativas, a verificação de validade do grafo de dependências também pode ser necessária durante a materialização AXML. Nesta tese, para otimizar a materialização de documentos AXML, pressupõe-se que os grafos de dependências são válidos.

4.4.4 ELIMINAÇÃO DE REDUNDÂNCIAS

A especificação de um documento AXML pode conter dependências de invocação redundantes. Por exemplo, na Figura 12, a aresta sc4→sc5 pode ser suprimida sem alterar o teor das restrições de invocação do grafo, já que sc4→*sc5 (de sc4→sc7 e sc7→sc5). O grafo resultante é mostrado na Figura 14. Anteriormente, foi considerada a eliminação de redundâncias entre parâmetros não-concretos de uma chamada de serviço. Esta idéia pode ser estendida para o documento AXML inteiro, baseada no conceito de equivalência entre grafos.

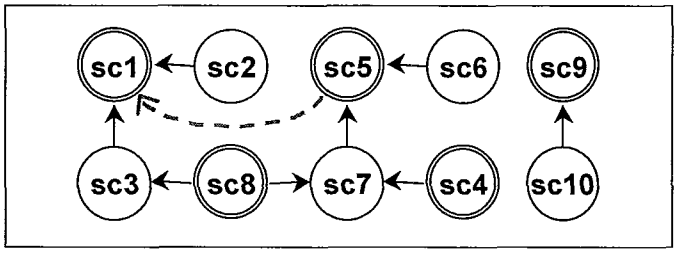


Figura 14. Grafo de dependências sem redundâncias.

Nesta tese, considera-se a equivalência entre grafos decorre do conceito de bissimilaridade, como em SUCIU (2002). Esta relação é baseada em regras de transformação que podem reduzir certos grafos a uma estrutura comum. Para grafos de dependências, a bissimilaridade se baseia na eliminação de dependências redundantes.

DEFINIÇÃO 11: Seja o grafo de dependências Δ e os nodos v_i e v_j em G. Se ambos $v_i \rightarrow v_j$ e $v_i \rightarrow^* v_j$ pertencem a E, então a aresta $v_i \rightarrow v_j$ é dita uma dependência redundante de Δ.

Portanto, ao se eliminar uma dependência redudante, o grafo resultante ainda mantém todas as restrições de invocação. A partir desta propriedade, tem-se o seguinte:

DEFINIÇÃO 12: Sejam os grafos de dependências Δ_a e Δ_b. Diz-se que Δ_a resume Δ_b, denotado por $\Delta_a \subset \Delta_b$, se e somente se:

- $V_a = V_b$, $\otimes_a = \otimes_b$, $\vec{E}_a = \vec{E}_b$, $\varepsilon_a = \varepsilon_b$ e $E_a \subset E_b$; e
- existe pelo menos um par de nodos v_i e v_j em V_b tal que $v_i \rightarrow v_j$ é uma dependência redundante de Δ_b e $v_i \rightarrow v_j$ não ocorre em E_a.

Com isso, considera-se que:

DEFINIÇÃO 13: Os grafos Δ_a e Δ_b são ditos bissimilares se $\Delta_a \subset \Delta_b$ ou $\Delta_b \subset \Delta_a$.

Um documento AXML pode conter diversas dependências redundantes. A eliminação dessas redundâncias é importante para evitar esforços desnecessários na otimização da materialização do documento. Vale ressaltar que o conceito de bissimulação entre grafos provê uma base formal para essa eliminação, gerando assim uma representação canônica para as restrições de invocação de um documento AXML. Tal representação consiste no grafo resumido mínimo do documento.

DEFINIÇÃO 14: Seja o grafo de dependências Δ. O grafo resumido mínimo de Δ é denotado por Δ^{MR}, tal que $\Delta^{MR} \subset \Delta$ e não há dependências redudantes em Δ^{MR}.

Outra propriedade interessante é que a ordem de eliminação das redundâncias de um grafo não interfere no formato do grafo resumido mínimo. Ou seja, pode-se garantir que, a partir de um grafo contendo dependências redundantes, existe somente um grafo resumido mínimo (a prova é dada na Proposição 1 do Anexo I). A Figura 14 mostra o grafo resumido mínimo do documento *SwapWorkspace*.

Redundâncias decorrem de dependências compartilhadas, devidas somente ao uso de parâmetros não-concretos. Ou seja, o grafo de dependências de um documento AXML contendo apenas parâmetros concretos é resumido ao mínimo por definição. No caso geral, a obtenção do grafo resumido mínimo possui complexidade de tempo $O(|V^3|)$, baseada no cálculo do fechamento transitivo do grafo. Contudo, em documentos AXML, pode-se verificar que esse limite é reduzido a $O(|\otimes| \times |V|^2)$, pois apenas chamadas de serviços com *fanOut* ≥ 1 podem originar arestas redundantes no grafo de dependências.

Nesta tese, salvo se expresso o contrário, considera-se que os grafos de dependência são resumidos ao mínimo.

4.4.5 PONTOS DE SAÍDA

Em um grafo de dependências, os nodos que não possuem arestas de saída (*i.e.*, com *fanOut* = *0*) são particularmente importantes para a materialização do documento AXML. Esses nodos são chamados de pontos de saída do grafo. Eles são usados na obtenção das árvores geradoras do grafo, para que o otimizador possa quebrar o problema de materialização em partes menores, reduzindo a complexidade global. Além disso, eles indicam pontos de finalização do processo de materialização. Após sua materialização e a ativação de suas chamadas colaterais, o processo de materialização encerra. Por exemplo, os nodos sc1, sc5 e sc9 são pontos de saída do grafo de dependências da Figura 14.

Cada grafo de dependências válido possui pelo menos um ponto de saída, como mostra o Lema 1 do Anexo I. Isso implica que, a despeito da complexidade resultante do uso de dependências compartilhadas e chamadas colaterais, o otimizador sempre é capaz de identificar um ponto de saída para avaliar um grafo de dependências válido. Assim, pressupondo que as execuções de serviços Web sempre encerram após um certo período de tempo e que respostas ativas não são infinitas, a materialização baseada em grafos válidos tem garantia de término (conforme a Proposição 2 do Anexo I). Vale mencionar que essa propriedade é baseada em uma interpretação instantânea do grafo de dependências, sem considerar as modificações que podem ocorrer devido a respostas ativas. A seguir, é abordada a evolução do grafo durante o processo de materialização AXML.

4.5 ATUALIZAÇÃO DINÂMICA DO GRAFO

Um sistema capaz de tratar documentos AXML pode permitir que serviços Web retornem dados contendo chamadas de serviços no resultado de uma invocação. Este artifício é útil em diversos cenários. Por exemplo, suponha que um serviço Web não dispõe de uma determinada informação que foi solicitada pelo usuário, mas conhece um outro serviço Web que pode fornecê-la. Em vez de retornar uma resposta vazia ou um erro, o serviço Web poderia dar uma resposta contendo outras chamadas de serviços. Com isso, o solicitante pode decidir se deve invocar (e quando) as indicações recebidas na resposta ativa.

No exemplo da aplicação *CurrencySwap*, na seção 2.2, suponha que o repositório de arquivos PDFs em P_4 está fora do ar. Para materializar o documento *SwapWorkspace* em um sistema com sítios *ActiveXML*, a invocação da chamada de serviço sc10 em P_4 poderia retornar o seguinte resultado:

```
<sc id="11" service="Ps2Pdf">
  <sc id="12" service="GetContractPS"/> </sc>
```

Então, para continuar a materializar o documento *SwapWorkspace*, o sítio mestre teria que invocar tanto sc11 como sc12, para obter os dados a serem passados como entrada de sc9 (no parâmetro "text"). Observe que esta resposta ativa pode demandar que o sítio mestre descobra informações sobre os sítios que provêm os serviços Web de sc11 e sc12.

Com respostas ativas, a materialização de documentos AXML pode ser dinamicamente distribuída, oferecendo aos sítios uma maior flexibilidade para colaboração. Fora do escopo de documentos AXML, o grande potencial do uso de respostas ativas no processamento de consultas é abordado por JIM e SUCIU (2001).

Entretanto, respostas ativas podem alterar significativamente um documento AXML, introduzindo vários problemas complexos no processo de materialização. Entre esses problemas, destacam-se os seguintes:

(i) o grafo de dependências precisa ser atualizado em tempo de execução com as novas chamadas de serviços, devendo continuar válido após sua atualização;

(ii) no sítio que recebe a resposta intencional, pode ser necessária a atualização de catálogos contendo informações sobre os serviços Web solicitados, abrangendo a localização de endereços e a aquisição de estatísticas e parâmetros de custo;

(iii) como a especificação AXML é alterada, envolvendo novos fluxos de dados e possivelmente outros provedores de serviços Web, decisões de otimização anteriores podem se tornar inválidas; e

(iv) alguns serviços Web podem retornar respostas ativas infinitas ou indesejadas.

Para tratar o problema do item (*iv*), MILO *et al.* (2003) propõem um mecanismo baseado em tipos de dados. Em MILO *et al.* (2003), respostas ativas devem ser recursivamente materializadas até que o documento AXML satisfaça um determinado tipo (previamente especificado pelo usuário) ou um ponto fixo seja atingido. Por outro lado, embora complementar à proposta de MILO *et al.* (2003), o foco desta tese é o desempenho da materialização de documentos AXML. Assim, são abordados especialmente os problemas relativos aos itens (*i*), (*ii*) e (*iii*). Em particular, esta seção apresenta algoritmos para tratar o item (*i*).

4.5.1 ALGORITMO BÁSICO DE ATUALIZAÇÃO

Respostas ativas permitem que um documento AXML "evolua" durante o processo de materialização. Conseqüentemente, essas respostas provocam operações de atualização no grafo de dependências. Vale lembrar que nodos intencionais permanecem inativos (ou com falha) após a invocação e que este estado só pode ser alterado pelo disparo de uma chamada colateral. Ou seja, mesmo sem receber respostas ativas, o grafo de dependências se modifica ao longo do processo de materialização. Porém, essas modificações não introduzem chamadas de serviços alienígenas no documento, como acontece com as respostas ativas.

DEFINIÇÃO 15: Seja $\langle D, \Rightarrow \rangle$ um documento AXML, seu grafo de dependências Δ e uma chamada de serviço v de Δ. Além disso, seja $\langle D_u, \Rightarrow_u \rangle$ um documento AXML retornado pela invocação de v, a ser inserido em $\langle D, \Rightarrow \rangle$. Se o grafo Δ_u obtido de $\langle D_u, \Rightarrow_u \rangle$ não é nulo, então Δ_u é dito uma resposta ativa de v.

A chamada de serviço cuja invocação resulta na resposta ativa é dita nodo de origem. Para atualizar o grafo de dependências, uma resposta ativa deve ser devidamente conectada a ele, preservando as restrições de invocação do nodo de origem em relação aos demais nodos do documento AXML.

DEFINIÇÃO 16: Uma operação de atualização u é expressa por $\langle \Delta, \Delta_u, v, \gamma \rangle$, onde Δ é um grafo de dependências contendo o nodo v, o grafo Δ_u é uma resposta ativa de v e o termo γ representa uma função que transforma Δ em um grafo $\Delta'=\langle G', \otimes', \vec{E}', \varepsilon' \rangle$, tal que $(V \cup V_u) \subseteq V'$ e $(E \cup E_u) \subseteq E'$.

Ao conectar uma resposta ativa no grafo de dependências, são criadas arestas para representar as novas restrições de invocação do documento. Nesta tese, verificou-se que algumas heurísticas são úteis para reduzir a complexidade da atualização do grafo. Baseado em algumas propriedades dos documentos AXML, as seguintes premissas são consideradas:

- As chamadas de serviços de uma resposta ativa não são afetadas pelas arestas colaterais do grafo. Isso ocorre porque chamadas colaterais apontam para referências específicas do documento AXML. Portanto, não é possível referenciar as chamadas de uma resposta ativa antes de sua inserção no grafo. Além disso, chamadas colaterais representam novas invocações de uma chamada de serviço, independente do resultado de invocações prévias. Portanto, as chamadas de uma resposta ativa não herdam os relacionamentos colaterais do seu nodo de origem, nem são referenciadas por nodos pré-existentes;

- Os parâmetros não-concretos contidos em uma resposta ativa não referenciam nodos do documento AXML, pois serviços Web geralmente desconhecem o conteúdo do documento e estes parâmetros envolvem especificações relativas ao contexto onde estão inseridos;

- Analogamente, as chamadas colaterais de uma resposta ativa referem-se apenas aos novos nodos, isto é, aos nodos contidos na resposta ativa; e

- Os nodos da resposta ativa não dependem dos parâmetros intencionais do nodo de origem, já que o resultado de uma invocação ou substitui totalmente (*i.e.*, toda a sub-árvore) ou é inserido como irmão do nodo de origem no documento AXML.

Estas heurísticas indicam que respostas ativas são fracamente acopladas ao grafo de dependências. Logo, apenas as arestas simples saindo do nodo de origem precisam

ser consideradas para conectar as chamadas de serviços de uma resposta ativa. Basicamente, a função γ realiza o seguinte algoritmo:

1) acrescenta os nodos da resposta ativa ao conjunto de nodos do grafo de dependências, preservando seus estados;
2) acrescenta as arestas (simples e colaterais) entre nodos da resposta ativa;
3) se o nodo de origem for uma chamada de primeiro nível, acrescenta os nodos de primeiro nível da resposta ativa ao conjunto de nodos persistentes do grafo;
4) cria arestas simples ligando cada chamada de primeiro nível da resposta ativa às chamadas de serviços que dependem do nodo de origem no grafo; e
5) recalcula os parâmetros não-concretos do documento AXML.

Esses passos necessários são detalhados no algoritmo *Update* do Anexo I.

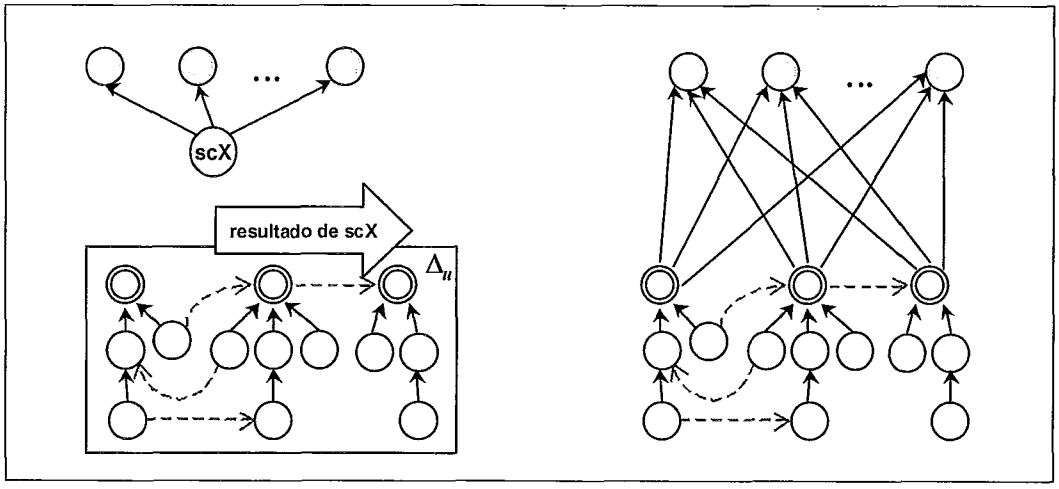


Figura 15. Atualização de um grafo de dependências por uma resposta ativa.

A atualização de um grafo de dependências é ilustrada na Figura 15. O principal passo consiste em propagar as restrições de invocação do nodo de origem às chamadas de serviços da resposta ativa. No caso geral, essa propagação pode se tornar muito custosa, dependendo do *fanOut* do nodo de origem e do número de chamadas de primeiro nível da resposta ativa.

4.5.2 ATUALIZAÇÃO BASEADA EM NODOS *PIPE*

Apesar das simplificações discutidas na seção anterior, atualizar um grafo de dependências pode envolver a criação de várias arestas para ligar os novos nodos. Para estabelecer limites na criação de arestas e melhorar o desempenho deste procedimento, esta tese propõe o uso de um nodo especial chamado *pipe*.

Nodos do tipo *pipe* visam simplificar a atualização de grafos de dependências, concentrando as arestas que conectam os nodos de um grafo aos nodos de uma resposta ativa, conforme ilustrado na Figura 16. Um nodo *pipe* é transparente em relação ao conteúdo de suas entradas, pois não produz nem elimina dados. Ele faz referência a um serviço Web genérico (*i.e.*, que pode ser executado potencialmente por qualquer sítio), cuja semântica é simples: ele recebe o resultado das invocações de suas dependências e os passa para seus dependentes, tal que cada dependente do nodo *pipe* recebe os resultados de todas as dependências.

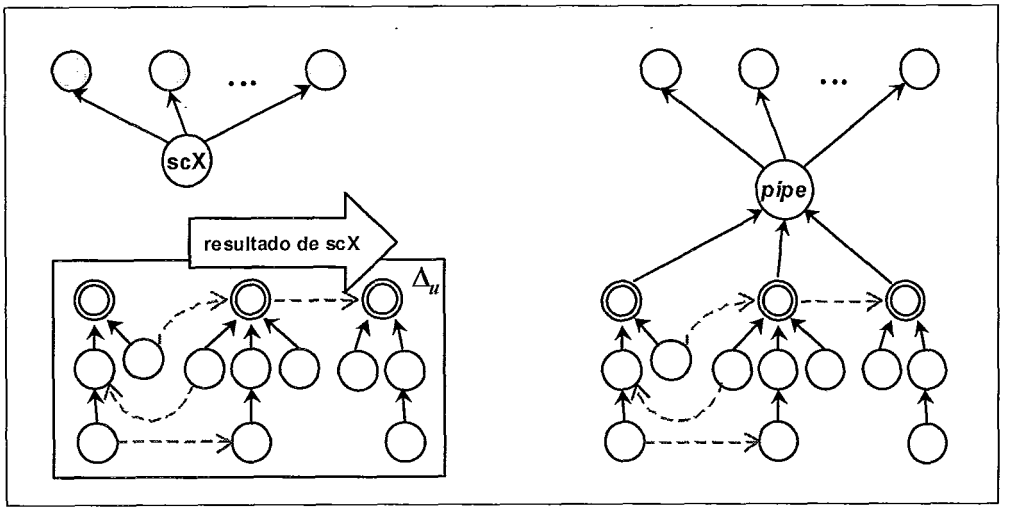


Figura 16. Atualização do grafo de dependências com nodos *pipe*.

Vale ressaltar que se uma resposta ativa Δ_u é conectada sem nodos *pipe*, então o número máximo de arestas para ligá-la ao grafo de dependências é dado por:

$$fanOut(v) \times |\xi_u| \ . \qquad (2)$$

onde v é o nodo de origem. Já com nodos *pipe*, esse número é se limita a:

$$fanOut(v) + |\xi_u| \ . \qquad (3)$$

Assim, há uma redução significativa no número de arestas criadas na atualização do grafo de dependências. Esse limite também beneficia a gerência da memória principal necessária para armazenar o grafo.

O uso de nodos *pipe* na atualização de um grafo de dependências é particularmente útil quando o nodo de origem é uma dependência compartilhada (*i.e.*, quando o *fanOut* do nodo de origem é maior que 1). Além disso, também é interessante usar operadores *pipe* quando se faz necessário relaxar as premissas de acoplamento fraco da resposta ativa. Por exemplo, caso se queira determinar que os nodos da resposta ativa herdem as chamadas colaterais do nodo de origem. Vale destacar que, como nodos *pipe* são inócuos, um grafo de dependências atualizado com esses nodos é equivalente ao grafo atualizado com o algoritmo básico. Ou seja:

Em um grafo de dependências Δ, a substituição de qualquer aresta $v_i \rightarrow v_j$ em E pelo caminho $v_i \rightarrow pipe \rightarrow v_j$ não altera as restrições de invocação entre as chamadas de serviços de Δ.

Nodos *pipe* são inspirados nas arestas "*epsilon*", que foram introduzidas por SUCIU (2002) para simplificar tanto a representação como as operações de transformação de grafos complexos.

4.5.3 VALIDADE DO GRAFO ATUALIZADO

Para atualizar corretamente um grafo de dependências com uma resposta ativa, é necessário verificar se o grafo resultante contém ciclos maliciosos. Ou seja, uma operação de atualização só deve ser aceita se o grafo resultante for válido.

DEFINIÇÃO 17: Uma operação de atualização u é dita correta se e somente se o grafo resultante Δ' for válido.

Como uma resposta ativa é fracamente acoplada ao grafo de dependências, uma operação de atualização conecta os novos nodos tal que: eles não dependem das

chamadas de serviços do grafo; e eles não disparam (nem são disparados por) chamadas colaterais do grafo original. Com isso, pode-se verificar que os nodos de uma resposta ativa não contribuem para formar ciclos maliciosos com os nodos do grafo de dependências. Portanto:

A partir de um grafo de dependências válido, para uma operação de atualização ser considerada correta, apenas a validade da resposta ativa precisa ser verificada.

Este resultado é apresentado na Proposição 3 do Anexo I. Vale destacar que uma vantagem da abordagem proposta consiste em poder verificar se uma operação de atualização é correta antes mesmo que qualquer alteração seja feita no documento AXML. Isto é possível pela análise do grafo de dependências.

4.5.4 CONTROLE DAS ATUALIZAÇÕES DO GRAFO

Outra questão importante na atualização de um grafo de dependências com respostas ativas diz respeito ao recálculo dos parâmetros não-concretos. Quando um documento AXML é alterado, a função Φ deve ser re-avaliada para as expressões XPath que definem parâmetros não-concretos. Geralmente, essa re-avaliação é bastante custosa, podendo se tornar inviável caso atualizações freqüentes sejam necessárias.

Uma alternativa para reduzir o tempo gasto recalculando os parâmetros não-concretos de um documento AXML consiste em realizar uma análise tolerante. A idéia é fazer a re-avaliação somente após um determinado período, definido por algum critério como o número de invocações de serviços ou de respostas ativas. Para formalizar esta idéia, é necessário antes definir como ocorre o processo de materialização de um documento AXML, baseado nas modificações causadas no grafo de dependências.

DEFINIÇÃO 18: Em um documento AXML $\langle D, \Rightarrow \rangle$, uma seqüência de invocações $I = \lfloor v_1, \ldots, v_\eta \rfloor$ representa uma seleção sucessiva de nodos prontos de SC_D, com $\eta \geq 0$, tal que:

- cada nodo selecionado é invocado e a última invocação é a de v_η;
- para quaisquer nodos v_i e v_j em I, se $v_i \rightarrow v_j$ em $\langle D, \Rightarrow \rangle$, então $i < j$; e

- para quaisquer nodos v_i e v_j em I, se $v_i \Rightarrow v_j$ em $\langle D, \Rightarrow \rangle$, então $j > i$ e os únicos nodos entre v_i e v_j em I são as dependências de v_j e suas respectivas chamadas colaterais.

O termo η indica a duração de I, que pode ser infinita.

Observe que os nodos são tratados em uma determinada ordem, respeitando as restrições de invocação do documento AXML. A duração de uma seqüência de invocações é infinita quando o documento AXML possui ao menos uma chamada de serviço que sempre retorna uma resposta ativa recursiva (*i.e.*, contendo uma nova chamada para o mesmo serviço Web). Também vale destacar que o conjunto SC_D é dinâmico na Definição 18, pois pode ser acrescido com nodos de respostas ativas ao longo da seqüência de invocações. Além disso, para uma seqüência de invocações com duração n, denota-se por Δ_x o grafo de dependências obtido de $\langle D, \Rightarrow \rangle$ após a invocação de v_x, com $1 \leq x \leq \eta$. Analogamente, o grafo de dependências de $\langle D, \Rightarrow \rangle$ obtido antes do início da seqüência de invocações é indicado por Δ_0.

Mesmo que não ocorram respostas ativas, o grafo de dependências é alterado ao longo de uma seqüência de invocações, para refletir as mudanças de estado das chamadas de serviços. A materialização de um documento AXML acontece da seguinte forma:

DEFINIÇÃO 19: Uma seqüência de invocações I materializa um documento AXML $\langle D, \Rightarrow \rangle$ se e somente se não há nodos prontos em Δ_η. Em particular, diz-se que I é efetiva se $\varepsilon_\eta(v) = \textit{inativa}$ para todo nodo v em Δ_η.

Como um grafo de dependências determina uma ordem parcial na invocação de suas chamadas de serviços, um documento AXML pode ser materializado por diferentes seqüências de invocações equivalentes. Por exemplo, o documento *SwapWorkspace* pode ser materializado por seqüências iniciadas por sc2, sc4, sc6, sc8 ou sc10. Neste caso, o conceito de equivalência é baseado no grafo de dependências resultante e nas chamadas de serviços invocadas.

DEFINIÇÃO 20: Duas seqüências de invocações I_a e I_b são equivalentes se:

- $\Delta_{\eta a} = \Delta_{\eta b}$, onde η_a e η_b indicam, respectivamente, as durações de I_a e I_b; e
- o número de ocorrências de cada chamada de serviço em I_a é exatamente igual em I_b, e vice-versa.

Uma alternativa para controlar as alterações sofridas pelo grafo de dependências durante uma seqüência de invocações consiste em definir fases de materialização que delimitem o impacto das eventuais respostas ativas.

DEFINIÇÃO 21: Sejam $\langle D, \Rightarrow \rangle$ um documento AXML e uma seqüência de invocações $I_{timeline}$ que materializa $\langle D, \Rightarrow \rangle$. Uma fase de materialização de $\langle D, \Rightarrow \rangle$ em $I_{timeline}$ é denotada pela expressão $\varphi_{D,Itimeline} = \langle \Delta_0, \wp, IT, U, \beta \rangle$, onde:

- Δ_0 é o grafo de dependências obtido de $\langle D, \Rightarrow \rangle$ antes de v_1 ser invocado;
- $\wp$ é o critério de duração da fase, representando uma combinação lógica de predicados sobre propriedades de $\varphi_{D,Itimeline}$, o qual é avaliado a cada invocação de v_x em $I_{timeline}$, com $1 \leq x \leq \eta_{timeline}$, até se tornar verdadeiro;
- IT é a trilha de invocações da fase, definida por uma seqüência de invocações $[v_1, \ldots, v_{\eta\phi}]$ ocorridas até $\wp$ se tornar verdadeiro, tal que $IT \subseteq I_{timeline}$. O número η_ϕ indica a duração de $\varphi_{D,Itimeline}$, com $1 \leq \eta_\varphi \leq \eta_{timeline}$;
- U consiste em uma seqüência ordenada e possivelmente vazia de operações de atualização provocadas por IT em Δ_0; e
- β denota o acumulador da fase, que consiste em uma função sobrejetora de nodos de IT em nodos de SC_D, tal que $\beta(v_x)$ representa o conjunto de nodos que foram adicionados a SC_D por respostas ativas decorrentes das invocações de v_1 até v_x, incluindo v_x, com $1 \leq x \leq \eta_\varphi$.

Uma fase é dita incompleta enquanto $\wp$ for falso e $IT \neq I_{timeline}$. Caso contrário, a fase é completa. Por fim, a seqüência $I_{timeline}$ é dita a guia de $\varphi_{D,Itimeline}$.

Em linhas gerais, uma fase de materialização representa um período (medido em número de chamadas de serviços) no qual o conteúdo de um documento AXML é

parcial ou totalmente materializado. O critério de duração de uma fase pode ser definido por diversas métricas, tais como o número máximo de invocações em *IT*, o número máximo de nodos ativos no documento AXML e o tempo máximo gasto na execução das chamadas de serviços. Por exemplo, um critério de duração pode ser dado pela expressão "$|IT| = 5$", indicando que a fase de materialização será completa quando a trilha de invocações tiver cinco chamadas de serviços.

A validade do grafo de dependências também deve respeitada ao longo de uma fase de materialização. Isso implica que as operações de atualização da fase devem ser corretas. Vale destacar que, já que essas operações são consideradas em seqüência em uma fase, basta garantir que cada operação seja correta para que o grafo de dependências final seja válido.

Baseado em fases de materialização, pode-se determinar intervalos para re-avaliar parâmetros não-concretos, postergando a análise de Φ para quando houver um número relevante de alterações. Vale relembrar que Φ identifica as chamadas de serviços que contribuem para a expressão XPath de um parâmetro de entrada. Para postegar a análise de Φ, deve-se eliminar o passo 5 do algoritmo básico que implementa a função γ nas operações de atualização, as quais passam a ser tratadas como tolerantes. Com isso, além do critério de duração de uma fase de materialização, é necessário estabelecer um mecanismo de controle para disparar periodicamente Φ.

DEFINIÇÃO 22: Uma fase de materialização tolerante é denotada por $\langle \varphi_{D,Itimeline}, len \rangle$, onde:

- todas as operações de atualização em U são tolerantes; e
- *len* é um critério de tolerância definido por uma combinação lógica de predicados da forma "$|A|$ *operador valor*", tal que A está em $\{SC_D, IT, U, \beta\}$, o termo *operador* está em $\{=, >, <, \geq, \leq\}$ e *valor* é um número inteiro.

Além disso, após a invocação de cada nodo v_x em *IT*, com $1 \leq x \leq \eta_\varphi$, a re-avaliação dos parâmetros não-concretos de Δ_x é disparada se *len* for verdadeiro ou $x = \eta_\varphi$.

O critério de tolerância é limitado a comparações com números de chamadas de serviços devido à ênfase da análise tolerante neste tipo de métrica, que é essencial para controlar o volume de alterações a serem feitas no grafo de dependências inicial.

Uma vantagem importante da análise tolerante é que, a cada ponto de re-avaliação, a função Φ precisa considerar apenas os nodos acumulados (*i.e.*, em β) para detectar novos parâmetros não-concretos. Porém, para que todas as dependências de invocação sejam respeitadas entre pontos de re-avaliação, pressupõe-se que a função Φ é sempre avaliada quando se invoca uma chamada de serviço baseada em expressões XPath. Deste modo, nenhuma dependência de invocação é ignorada, mesmo que grafo esteja desatualizado devido à análise tolerante.

4.5.5 COMENTÁRIOS

A representação das restrições de invocação entre as chamadas de serviços é uma questão crucial na materialização de documentos AXML. Em particular, tal representação permite verificar propriedades importantes, como a garantia de término do processo de materialização. Ela contribui também para um controle efetivo da evolução de documentos AXML. Vale destacar que o modelo canônico proposto é baseado em uma estrutura amplamente conhecida (*i.e.*, grafos). Com isso, as idéias básicas da estratégia de otimização desta tese podem ser aplicadas em um contexto mais abrangente, como em sistemas para gerência de *workflows*. Aliás, o modelo proposto provê um suporte formal que pode ser explorado por outras abordagens para a otimização da materialização AXML.

Também é importante ressaltar que a materialização de um documento AXML geralmente não é um processo sequencial, pois ela pode ser quebrada em um conjunto de seqüências de invocações tal que algumas delas são realizadas em paralelo. Isso ocorre porque em um grafo de dependências existem chamadas de serviços cujas invocações são independentes entre si.

4.6 Modelo P2P para Materialização AXML

O grafo de dependências de um documento AXML consiste em uma representação de alto nível das restrições de invocação entre chamadas de serviços, a qual não contém informações sobre o uso de recursos computacionais. Entretanto, para materializar um documento AXML, deve-se determinar que sítio executará cada chamada de serviço. Com a possibilidade de colaboração P2P, esse mapeamento de chamadas de serviços em sítios também deve definir quem controlará a materialização de cada parte do grafo de dependências. Nesta seção, é proposto um modelo P2P para criar instâncias da materialização de documentos AXML a partir do grafo de dependências. São descritos os principais participantes na materialização AXML e como eles podem colaborar entre si em um cenário P2P.

4.6.1 Especificações Abstratas de Documentos AXML

A forma mais básica de especificar um documento AXML consiste em determinar um endereço específico para cada chamada de serviço Web, informando uma URL, a porta de acesso e as demais informações necessárias à invocação do serviço Web, conforme definido pelo protocolo SOAP (SOAP, 2003). Neste caso, cabe ao usuário localizar todos os sítios que irão executar cada chamada de serviço do documento AXML. Esta abordagem é notoriamente trabalhosa para o usuário, principalmente porque sistemas P2P são geralmente compostos por arranjos dinâmicos e complexos de muitos sítios. Além disso, ela faz com que o documento AXML fique "engessado", tornando sua materialização mais vulnerável a falhas nas invocações de serviços durante o processo de materialização.

Em uma abordagem mais flexível, pode-se utilizar referências abstratas para identificar os serviços Web requisitados por um documento AXML. Tais referências são baseadas em alguma ontologia de serviços Web como, por exemplo, a OWL-S (OWLS, 2004). Referências abstratas costumam corresponder a entradas de um catálogo de serviços ou de um repositório UDDI (UDDI, 2004), onde elas são associadas aos sítios que provêm os respectivos serviços Web. Além disso, geralmente serviços Web são providos por vários sítios em um sistema P2P. Assim, a escolha do melhor provedor

(em termos de desempenho) para executar cada chamada de serviço de um documento AXML deve ser automática.

Vale destacar que, mesmo quando são usadas especificações concretas, sítios podem colaborar para materializar um documento AXML por meio da delegação de tarefas. Logo, faz-se necessária uma estratégia que permita distribuir eficientemente o processo de materialização AXML no sistema P2P. Nesta tese, considera-se um conjunto infinito **P** de nomes de sítios do sistema.

DEFINIÇÃO 23: A lista de vizinhos de um sítio P é denotada por $\check{N}_P$ e representa o conjunto de sítios habilitados a colaborar com P na materialização de documentos AXML, tal que $\check{N}_P \subseteq \mathbf{P}$.

Note $\check{N}_P$ que pode variar ao longo do funcionamento do sítio P, devido ao dinamismo do sistema P2P.

4.6.2 USO DE SERVIÇOS WEB EQUIVALENTES

A evolução da Web semântica tem fomentado a orquestração de serviços Web em diversos cenários distribuídos, tais como em sistemas P2P (ABITEBOUL *et al.*, 2004e, BRADLEY e MAHLER, 2004, FERREIRA *et al.*, 2003) e em grades computacionais (BLYTHE *et al.*, 2004, CHEUNG *et al.*, 2004, HUANG *et al.*, 2005, VERMA *et al.*, 2005, ZENG *et al.*, 2004). Nesses sistemas, as aplicações distribuídas são definidas por especificações abstratas e executadas em um ambiente baseado em serviços Web equivalentes. Para isso, utiliza-se um catálogo de serviços Web e algum mecanismo que permita a seleção de serviço a partir de um conjunto de critérios (BURSTEIN *et al.*, 2002, COSTA *et al.*, 2004, MEDJAHED e BOUGUETTAYA, 2005).

No contexto de documentos AXML, referências abstratas nas chamadas de serviços podem ser mapeadas em um conjunto de endereços de serviços Web equivalentes. Assim, cabe ao otimizador selecionar o melhor endereço para cada referência. Porém, vale destacar que a tradução dessas referências não recupera

composições de serviços. Ou seja, nesta tese se pressupõe que cada referência sempre é traduzida em somente um endereço de serviço Web.

Um caso simples de equivalência de serviços existe quando serviços Web são arbitrariamente replicados no sistema P2P. Em geral, a replicação de dados e/ou serviços Web é usada em cenários distribuídos para aumentar o desempenho e a confiabilidade de um sistema (ALPDEMIR *et al.*, 2004, BLYTHE *et al.*, 2004). Na plataforma *ActiveXML*, a replicação de serviços é particularmente interessante para os serviços declarativos (*i.e.*, que são definidos por consultas). Isso ocorre porque qualquer sítio *ActiveXML* pode avaliar uma consulta e, portanto, se tornar um provedor do serviço Web correspondente. Para tanto, basta que o sítio possua os dados necessários à consulta. Um mecanismo para replicação de dados e serviços na plataforma *ActiveXML* é proposto por ABITEBOUL *et al.* (2003b).

Outra classe de serviços equivalentes são os serviços genéricos. Esses serviços costumam estar disponíveis em diversos sítios pois realizam tarefas genéricas, tais como criptografia e compressão de dados. Em geral, eles não alteram o conteúdo de suas entradas, mas sobretudo agem em algum aspecto não-funcional. Por exemplo, um serviço de criptografia altera a visibilidade dos dados.

Para apoiar a materialização de documentos AXML, sítios coletam informações sobre a distribuição de serviços Web equivalentes no sistema P2P (como proposto em ABITEBOUL *et al.*, 2004e). Tais informações não representam necessariamente o estado global do sistema, mas apenas a visão de um determinado sítio. Além disso, considera-se que em alguns casos o otimizador é capaz de replicar serviços declarativos ou genéricos durante o processo de materialização, visando melhorar o desempenho do sistema. Contudo, é bom lembrar que a replicação dinâmica de serviços Web pode aumentar substancialmente o número de alternativas de avaliação da materialização AXML. Logo, o otimizador deve estimar se o aumento do espaço de alternativas provocado pelo replicação de serviços Web torna o problema de materialização AXML muito complexo, o que comprometeria a avaliação.

Em RUBERG *et al.* (2004a), o modelo básico de documentos AXML foi estendido para permitir que chamadas de serviços aceitem o termo ***any*** para indicar o provedor de um serviço Web. Com este recurso, o usuário não precisa definir nenhum

provedor em particular para uma chamada de serviço, deixando que o otimizador faça a seleção baseada em critérios de desempenho. Nesta tese, também é considerado o símbolo ***unknown*** para indicar que o otimizador não dispõe localmente de nenhuma informação sobre os sítios que provêem um determinado serviço Web. mas que ainda pode tentar descobrí-la no sistema P2P. Com isso, tem-se que:

DEFINIÇÃO 24: O escopo de execução Le_v de uma chamada de serviço v é denotado por Le_v= ***any*** | ***unknown*** | $\{P_1, \ldots, P_n\}$, onde ***any*** indica que todos os sítios vizinhos podem executar v, o símbolo ***unknown*** denota que Le_v está indefinido e $\{P_1, \ldots, P_n\}$ consiste em um conjunto finito de sítios em **P** que provêem o serviço de v, com $n \geq 0$.

Observe que $Le_v=\{\}$ significa que o otimizador não conseguiu descobrir nenhuma informação sobre os sítios que provêem o serviço Web do nodo v. Vale mencionar também que o provedor de uma chamada de serviço de um documento AXML não é necessariamente um vizinho do sítio mestre. Por exemplo, na Figura 4, o sítio mestre do documento *SwapWorkspace* é P_1. Seja $\check{N}_{P1} = \{P_2, P_4\}$. Neste caso, pode-se notar que o sítio P_3 é um provedor da chamada de serviço sc1, embora ele não esteja em $\check{N}_{P1}$. Na materialização AXML, o executor de cada chamada de serviço é selecionado a partir do respectivo escopo de execução.

Como sítios podem entrar ou sair do sistema aleatoriamente, o escopo de execução de uma chamada de serviço representa sobretudo uma fotografia do sistema, com a perspectiva de um determinado sítio. Assim, mesmo se o escopo de execução de uma chamada de serviço esteja vazio, o sítio ainda pode tentar adquirir novas informações posteriormente, esperando por alguma mudança no sistema. Além disso, um sítio também pode solicitar a um vizinho que localize os provedores de uma chamada de serviço. A seguir, são discutidos os principais mecanismos de colaboração entre sítios.

4.6.3 ALTERNATIVAS DE COLABORAÇÃO P2P

A materialização de um documento AXML é inerentemente distribuída, pois os provedores dos serviços Web solicitados geralmente estão espalhados na rede. Em uma

arquitetura P2P orientada a serviços, várias outras formas de colaboração também podem ser exploradas, com destaque para os seguintes cenários:

- sítios podem colaborar para localizar provedores de serviços Web;
- sítios podem delegar o controle da execução de partes da materialização de um documento AXML para outros sítios; e
- analogamente, sítios podem delegar a otimização de partes da materialização AXML.

Basicamente, a materialização AXML ocorre em um sistema P2P por meio da troca de mensagens entre sítios. Nesta tese, a base da interação entre sítios é o plano de materialização, que permite controlar o processo de materialização AXML. Um plano de materialização determina como cada chamada de serviço do documento AXML será invocada, podendo ser dividido e distribuído entre sítios. Além disso, sítios que colaboram entre si podem rever algumas decisões de otimização de planos recebidos, para então redistribuí-los entre outros sítios. Assim, o processo de materialização pode ser propagado no sistema P2P de forma descentralizada. Planos de materialização são formalmente definidos no Anexo I.

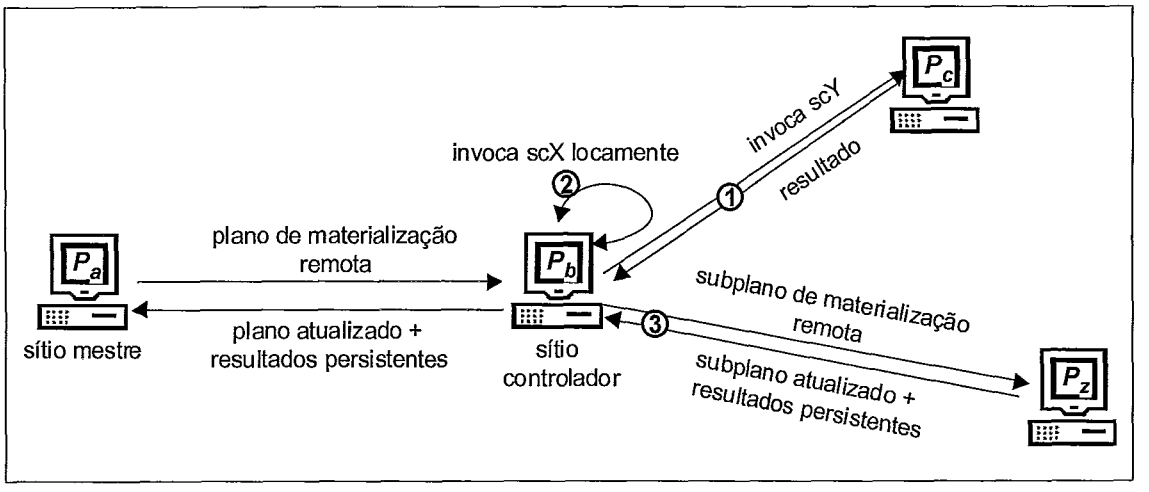


Figura 17. Delegação de subplanos na materialização de documentos AXML.

Enquanto a localização de serviços Web é geralmente uma alternativa comum de colaboração P2P, delegar o controle da execução de tarefas requer mecanismos mais sofisticados. Em um documento AXML, delegar uma chamada de serviço consiste em

outorgar tanto sua invocação como o recebimento de seu resultado a um outro sítio (*i.e.*, que não seja o sítio mestre). Para isso, o sítio mestre envia um plano de materialização remota contendo a chamada de serviço, seus parâmetros de entrada (possivelmente incluindo suas dependências de invocação) e sua chamada colateral. Por sua vez, o sítio outorgado é dito o controlador da chamada de serviço. Ele deve executar esse plano e retornar os resultados ao sítio mestre. Porém, apenas os resultados persistentes do plano de materialização remota precisam ser retornados ao sítio mestre. Esses resultados são embutidos no próprio plano, cuja versão atualizada após a execução também é retornada ao sítio mestre. Deste modo, caso o controlador não tenha conseguido executar todo o plano, ele ainda pode retorná-lo parcialmente materializado. Esse processo é ilustrado na Figura 17.

Quanto às chamadas de serviços de um plano de materialização remota, conforme mostra a Figura 17, dependendo da distribuição de serviços Web no sistema P2P, o controlador pode: 1) invocá-las disparando a execução em outros sítios; 2) invocá-las localmente, caso seja o provedor do serviço Web solicitado; ou 3) delegá-las a outros sítios.

Nesta tese, pressupõe-se que sítios são autônomos. Portanto, o sítio mestre não pode impor uma delegação transitiva aos sítios. Ou seja, ele não pode determinar que um controlador delegue parte de um plano de materialização remota, pois essa decisão é tomada em cada sítio. Assim, o resultado de um plano de materialização remota é sempre enviado para o sítio que o delegou.

DEFINIÇÃO 25: O escopo de delegação Ld_v de uma chamada de serviço v é representado por Ld_v = ***any*** | $\{P_1, \ldots, P_n\}$, onde ***any*** indica que todos os sítios em $\check{N}$ podem invocar v e $\{P_1, \ldots, P_n\}$ é um conjunto finito de sítios em $\mathbf{P}$ que podem invocar v, com $n \geq 0$.

Em geral, o otimizador pode considerar qualquer sítio no sistema para delegar alguma materialização AXML, desde que tal sítio possua as devidas permissões. Para decidir sobre a delegação das chamadas de serviços de um documento AXML, o otimizador leva conta principalmente o impacto da colaboração no desempenho do sistema. Em particular, tal decisão é fortemente influenciada pelos fluxos de dados entre

as chamadas de serviços. Por exemplo, a delegação é interessante na materialização dos nodos sc9 e sc10 do documento *SwapWorkspace*, pois permite explorar enlaces de comunicação mais rápidos entre os sítios P_4 e P_5.

Vale mencionar que a principal motivação da delegação de chamadas de serviços na materialização AXML é semelhante à idéia dos algoritmos "*edge-zeroing*", que são utilizados para o escalonamento de tarefas em computação paralela (KWOK e AHMAD, 1999). A partir de um grafo de tarefas interligadas por fluxos de dados, esses algoritmos analisam as arestas com os maiores custos de comunicação e tentam definir grupos a serem executados em um mesmo processador. Entretanto, conforme é mostrado em RUBERG *et al.* (2004a), na materialização de documentos AXML, os custos de comunicação não são irrisórios se duas chamadas de serviço relacionadas por uma dependência de invocação são executadas em um mesmo sítio. Isso ocorre porque o protocolo SOAP geralmente envolve operações bastante custosas, além da transferência do dados propriamente dita, tais como a serialização e o empacotamento dos dados a serem transferidos. Para decidir sobre as possíveis delegações de chamadas de serviços de um documento AXML, é necessário comparar as alternativas de avaliação por meio de métricas de custo adequadas.

Distribuir a otimização de um documento AXML consiste em um processo análogo à delegação de chamadas de serviços. Esse mecanismo de colaboração oferece uma abordagem descentralizada que contribui para lidar com sobrecargas de solicitações nos sítios e com a falta de informações de apoio. Além disso, se o problema de materialização AXML for muito complexo, o sítio mestre pode dividir o grafo de dependências sem fazer qualquer consideração sobre a localização dos serviços Web ou sobre o desempenho das alternativas de materialização. Então, as partes do grafo de dependências podem ser enviadas a outros sítios, que devem gerar os planos de materialização correspondentes e retorná-los ao sítio mestre para execução.

Tabela 3. Serviços para colaboração P2P na materialização de documentos AXML.

Serviço Web	**Descrição**
locate	Tenta descobrir o escopo de execução de um plano de materialização. Retorna um plano anotado com escopos de execução.

optimize	Gera um plano de materialização a partir de um plano incompleto.
submit	Recebe solicitações para executar um plano de materialização. Retorna o plano de materialização remota e seus resultados persistentes.

Para apoiar a colaboração P2P na plataforma *ActiveXML*, pressupõe-se que cada sítio deve prover os serviços básicos para colaboração P2P que são mostrados na Tabela 3. Observe que, ao receber uma solicitação para materializar um documento AXML, cabe ao sítio mestre decidir se outros sítios irão participar com a delegação, seja de chamadas de serviços ou da otimização. Por outro lado, já que sítios possuem perspectivas diferentes do sistema, cada controlador pode decidir re-otimizar um plano de materialização remota. Para evitar delegações infinitas, o plano de materialização remota inclui alguns limites de re-otimização, tais como o número máximo de delegações permitidas e a proveniência do plano (para evitar ciclos de delegação).

4.6.4 Instâncias da Materialização AXML

Planos de materialização representam alternativas de avaliação para a materialização de um documento AXML e dependem basicamente da configuração do grafo de dependências e das combinações dos escopos de execução e de delegação das chamadas de serviços. Uma instância de materialização AXML consiste na realização de um determinado plano de materialização em um sistema P2P. Neste contexto, os elementos básicos de um sistema P2P são:

- os sítios, que representam agentes unicamente identificados e conectados entre si através de uma rede P2P;
- os serviços Web, ou operações que os sítios podem realizar; e
- os documentos AXML, que são armazenados nos repositórios de cada sítio e apontam para os serviços Web do sistema.

Nesta tese, considera-se que sítios são heterogêneos e interconectados por diferentes enlaces de comunicação.

A execução de um serviço Web pode demandar parâmetros de entrada e possui um estado final (*e.g.*, "successo" ou "falha"). Invocar um serviço Web dispara sua execução e consiste em um evento discreto que permite: (*i*) o fluxo dos parâmetros de entrada para o sítio que vai executar o serviço; (*ii*) a execução propriamente dita do serviço; e (*iii*) a transferência dos resultados da execução para o sítio que solicitou o serviço. Em geral, chamadas de serviços são invocadas pelo sítio mestre (*i.e.*, que armazena o documento AXML). Porém, essas invocações podem ser delegadas, individualmente ou em conjunto, a outros sítios por meio de colaboração P2P. Vale lembrar que, para ser invocada, uma chamada de serviço precisa dispor de todas as informações necessárias para identificar o serviço Web solicitado. Logo:

DEFINIÇÃO 26: Seja uma chamada de serviço v em um sítio P_v. Uma invocação de v é denotada pela expressão $\langle v, Pe, Pc, status \rangle$, onde:

- o escopo de execução Le_v não é vazio nem indefinido;
- Pe e Pc são, respectivamente, o executor e o controlador de v, tal que $Pe \in Le_v$ e $Pc \in (\{P_v\} \cup Ld_v)$; e
- o termo *status* representa o estado da execução de v, tal que $status \in \{sucesso, falha\}$.

Por exemplo, supondo que execuções de serviços são realizadas sem falhas na Figura 6(a), ocorrem as invocações $\langle$sc10, P_5, P_4, *sucesso*$\rangle$ e $\langle$sc9, P_4, P_4, *sucesso*$\rangle$.

Com isso, para representar o mapeamento da materialização de um documento AXML nos sítios que participam do processo, são consideradas seqüências de invocação concretas, como a seguir.

DEFINIÇÃO 27: Uma seqüência de invocações I é dita concreta se e somente se todas as invocações de I são da forma $\langle v_x, Pe_x, Pc_x, status_x \rangle$, para $1 \leq x \leq \eta$.

Quando há algum problema em uma invocação de serviço durante a materialização AXML, as chamadas que dependem desta invocação não podem ser invocadas (já que não podem se tornar prontas) e as respectivas falhas são informadas

ao usuário. Contudo, pressupõe-se que as demais chamadas de serviços do documento AXML são invocadas sempre que possível. Além disso, por simplicidade, considera-se que o usuário não tem interesse em desfazer uma chamada de serviço quando há falhas na materialização de um documento AXML. Isto é, o documento AXML não representa uma transação única.

4.7 CONSIDERAÇÕES FINAIS

Em sistemas P2P, a materialização de um documento AXML geralmente pode ser realizada por um grande número de seqüências de materialização equivalentes. Além disso, para cada seqüência de invocação, podem ser gerados diferentes planos de materialização, conforme as combinações possíveis dos escopos de execução e de delegação das chamadas de serviços. Vale ressaltar que o desempenho da materialização AXML pode variar muito entre planos de materialização equivalentes. Portanto, para materializar eficientemente um documento AXML, é necessário gerar, comparar e executar os melhores planos, considerando especialmente a volatilidade dos sítios em um sistema P2P.

A seguir, no Capítulo 5, é apresentada uma estratégia dinâmica para otimizar a materialização de documentos AXML em sistemas P2P.

Capítulo 5

UMA ESTRATÉGIA DINÂMICA E DESCENTRALIZADA PARA A MATERIALIZAÇÃO EFICIENTE DE DOCUMENTOS AXML

O cerne desta tese é apresentado neste capítulo, que trata de uma estratégia descentralizada e baseada em custos para otimizar a materialização de documentos AXML. A estratégia proposta explora a geração dinâmica de planos de materialização, considerando a complexidade e a volatilidade dos sistemas P2P. É apresentada uma álgebra orientada a serviços Web, cujos operadores contemplam a colaboração P2P para delegar tanto a execução como para a própria otimização de planos de materialização.

"In preparing for battle I have always found that plans are useless, but planning is indispensable."

Dwight D. Eisenhower (1890-1969)

5.1 INTRODUÇÃO

A materialização de documentos AXML consiste em um processo bastante complexo, cujo desempenho é influenciado por um grande número de fatores. Neste contexto, a aplicação de técnicas de otimização enfrenta três grandes dificuldades: a distribuição inerente das execuções de serviços Web em um sistema heterogêneo; o número de planos de materialização equivalentes para um único documento AXML, que costuma implicar em enormes espaços de busca quando se considera a delegação de tarefas; e o forte dinamismo dos sistemas P2P, que torna pouco previsível o comportamento dos sítios. Para lidar com essas questões, o otimizador precisa ser dinâmico e capaz de explorar a colaboração P2P em uma abordagem descentralizada. Esses são os requisitos que motivaram a estratégia de otimização proposta nesta tese.

A necessidade de adaptação ao dinamismo dos sistemas P2P atinge vários aspectos da otimização, especialmente a localização de provedores de serviços Web, a escolha de sítios para colaboração e o cálculo de métricas de desempenho. Em métodos estáticos, planos são gerados completamente antes de serem executados. Assim, pressupõe-se que todas as informações necessárias à otimização (*e.g.*, as chamadas de serviços do documento AXML e seus relacionamentos, provedores de serviços Web, estatísticas e parâmetros de custos, etc.) estão disponíveis *a priori*. Uma abordagem estática é claramente inadequada para a materialização AXML, pois essas informações são voláteis e o otimizador deve reagir a mudanças no ambiente. Além disso, a adaptação baseada somente em re-otimizações também não é eficiente neste caso, pois pode provocar retrabalho em excesso.

Para adotar uma estratégia dinâmica sem se restringir a decisões demasiadamente locais, é necessário estabelecer critérios que permitam dividir adequadamente os planos gerados. Assim, passos de planejamento e execução podem ser intercalados, postergando as decisões do otimizador para o momento oportuno. Isso

também ajuda a reduzir a complexidade da busca por planos eficientes e favorece a descentralização. Entretanto, essa quebra deve preservar os relacionamentos entre os operadores do plano e permitir, sempre que possível, a obtenção de resultados parciais.

Vale mencionar que o objetivo da otimização de desempenho varia nas propostas da literatura, podendo ser baseado em tempo de resposta, balanceamento de carga, uso de recursos (*i.e.*, tempo total) ou custo financeiro (RUBERG, 2001). Contudo, em aplicações típicas da Web, o tempo de resposta é geralmente a métrica de maior impacto na percepção do usuário (CHERKASOVA *et al.*, 2003). Por isso, os padrões da Web costumam privilegiar a obtenção antecipada de resultados parciais, como ocorre no protocolo HTTP.

Esta tese propõe uma estratégia de otimização dinâmica e descentralizada para a materialização de documentos AXML. Em uma abordagem inovadora e abrangente, são tratados vários aspectos, com destaque para:

- a geração de planos de materialização a partir de grafos de dependências arbitrariamente complexos;
- uma álgebra de operadores para avaliação incremental e descentralizada de planos de materialização;
- um algoritmo para dividir adequadamente planos de materialização em partes menores, visando explorar o uso de heurísticas, apoiar uma otimização dinâmica e obter resultados parciais;
- métodos para enumerar e avaliar planos de materialização equivalentes, considerando as diferentes combinações dos escopos de execução e de delegação das chamadas de serviço de um grafo de dependências, além das principais heurísticas para agendamento de tarefas; e
- um modelo de custo para comparar alternativas de materialização AXML, contemplando uma ampla gama de fatores relevantes para o desempenho de planos de materialização, tais como os custos da delegação de chamadas de serviços, de execuções paralelas, da carga de processamento dos sítios e das operações típicas da invocação de serviços Web.

Em sistemas para a execução de *workflows* distribuídos, existem inúmeras heurísticas que visam auxiliar a otimização de desempenho. Todavia, convém lembrar que cada heurística geralmente se aplica a um cenário particular. A estratégia de otimização proposta desta tese permite que o otimizador escolha adequadamente o método de geração de planos para cada caso tratado, podendo até mesmo combinar os benefícios de duas ou mais heurísticas.

A estratégia proposta é realizada por um componente otimizador chamado **XCraft** (RUBERG e MATTOSO, 2007a, 2007b), baseado na plataforma *ActiveXML*. Pressupõe-se que o otimizador tem acesso ao repositório de documentos AXML e ao catálogo de serviços Web do sítio que o hospeda. O XCraft funciona em uma arquitetura orientada a serviços que apoia a colaboração P2P por meio da publicação de um conjunto de serviços Web (descritos na Tabela 3).

Este capítulo está organizado da seguinte forma. Na seção 5.2, é definido o problema de otimização tratado nesta tese. São descritos planos de materialização e delineados os principais limites do espaço formado pelos planos equivalentes de um documento AXML. Também é apresentada a métrica de desempenho utilizada pelo otimizador XCraft e é discutido o impacto das principais topologias de grafos de dependências. Em seguida, a seção 5.3 dá uma visão geral da estratégia de otimização dinâmica proposta. A conversão de grafos complexos em planos iniciais a partir de uma álgebra de operadores para materialização AXML é tratada na seção 5.4. Já na seção 5.5, é abordada a seleção de planos eficientes, a partir da geração e avaliação de alternativas de materialização AXML. Por fim, a 5.6 tece alguns comentários gerais. Os algoritmos e técnicas que constituem a estratégia proposta são detalhados no Anexo I.

5.2 PLANEJAMENTO DA MATERIALIZAÇÃO AXML

Para materializar eficientemente um documento AXML, o otimizador deve determinar um mapeamento entre as invocações de serviço do grafo de dependências e os sítios do sistema P2P, tal que o tempo decorrido para a obtenção do conteúdo do documento seja satisfatório. Entre os primeiros esforços da otimização, está a identificação dos escopos de execução e delegação das chamadas de serviços do

documento AXML. Dispondo dessas informações, o otimizador pode enumerar as alternativas de materialização para um documento AXML.

Nesta tese, o planejamento da materialização de um documento AXML consiste na geração e avaliação das alternativas de materialização, visando produzir um plano eficiente. O XCraft utiliza planos de materialização tanto para registrar essas alternativas e guiar o processo de otimização, como para controlar a realização da materialização.

DEFINIÇÃO 28: Um plano de invocação de uma chamada de serviço v consiste na tupla $\langle Pe, Pc \rangle$, onde Pc e Pe são sítios que podem, respectivamente, controlar a invocação de v e executar o serviço Web de v, tal que $Pc \in (Ld_v \cup Pm_v)$, $Pe \in Le_v$. e Pm_v é o sítio mestre de v. O termo $PI(v)$ representa o conjunto de todos os possíveis planos de invocação de v, de acordo com Ld_v e Le_v.

Basicamente, o otimizador tenta gerar um plano de invocação eficiente para cada chamada de serviço do documento AXML. Entretanto, como chamadas de serviços geralmente estão relacionadas entre si, o otimizador precisa avaliá-las em conjunto. No XCraft, um plano de materialização reúne informações sobre os planos de invocação das chamadas de serviços de um documento AXML. Cada chamada de serviço é tratada por um operador do plano, o qual indica como a chamada será avaliada. Esses operadores são descritos por uma álgebra orientada a serviços Web, conforme discutido na seção 5.4.2. Por ora, seja um conjunto de operadores algébricos representado por **A**.

DEFINIÇÃO 29: Um plano de materialização de um grafo de dependências Δ é denotado pela expressão $M_\Delta = \langle \Lambda, \alpha, L, >, Pm \rangle$, onde:

- Λ é um conjunto não-ordenado de árvores geradoras de Δ;
- α é um mapeamento que associa cada nodo de Δ a um operador em **A**;
- L é o escopo de realização de M_Δ, associando cada nodo v de Δ a um ou mais planos de invocação em $PI(v)$;
- o termo $>$ representa o agendamento de invocações de M_Δ e determina, para cada nodo v de Δ, uma ordem total nos filhos de v; e

- *Pm* é sítio mestre de M_Δ, ao qual devem ser enviados os resultados persistentes.

Diz-se que M_Δ é físico se e somente tanto L é total como ele mapeia cada nodo de Δ em exatamente um plano de invocação. Caso contrário, o plano M_Δ é dito abstrato.

Um plano de materialização é baseado no conjunto de árvores geradoras do grafo de dependências, pois essas estruturas são mais adequadas para uma abordagem de otimização descentralizada. A altura de um nodo em um plano de materialização é denotada por h e indica o comprimento do maior caminho entre o nodo e operadores que são folhas do plano. Uma variação desta propriedade é a menor altura de um nodo, relativa ao comprimento do menor caminho entre o nodo e as folhas do plano. Analogamente, a altura de um plano de materialização corresponde à maior altura dos operadores que enraizam as árvores geradoras.

5.2.1 Espaço de Planos Equivalentes

Devido à diversidade dos sistemas P2P e aos relacionamentos entre chamadas de serviços de um documento AXML, o número de planos de materialização equivalentes costuma ser muito grande. Considerando apenas as possíveis combinações dos escopos de execução das chamadas de serviços de um grafo de dependências Δ, o número de planos equivalentes é dado por:

$$\#escopoParcial(\Delta) = \prod_{\forall v, v \in V^*} |Le_v| \quad . \tag{4}$$

onde V^* indica o conjunto formado por todos os nodos de V mais os nodos referentes às chamadas colaterais de Δ. Neste caso, porém, o otimizador ignora as possibilidades de delegação de chamadas de serviços. Caso também seja considerado o escopo de delegação dos nodos, então o número de alternativas é:

$$\#escopoCompleto(\Delta) = \prod_{\forall v, v \in V^*} \left(|Le_v| \times \left(|Ld_v| + 1\right)\right) \quad . \tag{5}$$

Ou seja, são considerados os provedores de serviços do documento, os possíveis coordenadores e o sítio mestre do plano. No pior caso, o espaço de busca de complexidade exponencial da ordem de $O(s^{2|V^*|})$, em relação ao número s de sítios

distintos. Portanto, a delegação de chamadas de serviços aumenta significativamente a complexidade da geração de planos de materialização. Contudo, ela tem impacto significativo no desempenho da materialização AXML e deve ser explorada oportunamente.

Além da seleção de recursos, um plano deve determinar uma ordem para as chamadas de serviços do documento AXML, a partir das possíveis seqüências de invocações do grafo de dependências. Assim, o limite superior do número de planos físicos equivalentes é calculado por:

$$\#planos(\Delta) = |V^*|! \times \prod_{\forall v, v \in V^*} \left(|Le_v| \times \left(|Ld_v| + 1 \right) \right) \quad . \tag{6}$$

A busca de soluções eficientes neste espaço de planos equivalentes representa um problema bastante complexo, como geralmente ocorre na execução de *workflows* distribuídos. Portanto, na prática são necessárias heurísticas para lidar esse espaço de busca. Em particular, em RUBERG *et al.* (2004a), a heurística "Dividir para Conquistar" (D&C) é usada para dividir o grafo de dependências de um documento AXML em partes independentes. Com isso, o otimizador pode reduzir a complexidade da busca para:

$$\#planos^{D\&C}(\Delta) = \sum_{i=1}^{|\Lambda|} \#planos(\Delta_i) \quad , \tag{7}$$

onde $|\Lambda|$ indica o número de árvores independentes de Δ e $\#planos(\Delta_i)$ denota o número de planos equivalentes gerados a partir do subplano independente Δ_i. Observe que a heurística D&C não é ótima pois a avaliação de subplanos independentes pode envolver sítios em comum. Com isso, a avaliação de um subplano interfere no desempenho do outro e vice-versa.

5.2.2 Métrica de Otimização

A comparação do desempenho de planos equivalentes de um documento AXML é baseada no tempo de materialização. Essa métrica consiste no tempo de resposta decorrido entre o início do processo de materialização (*i.e.*, quando o conteúdo do

documento é solicitado) e a coleta dos resultados de todas as chamadas de serviços embutidas do documento.

Para estimar o tempo de materialização de um documento AXML, devem ser contabilizados diversos fatores, com destaque para os tempos gastos nas execuções de serviços Web, nas transferências de dados e nas comunicações decorrentes de colaboração P2P. O tempo de materialização é calculado pelo caminho crítico do grafo de dependências, que representa a seqüência de invocações com maior tempo de resposta. O cálculo do caminho crítico é estático quando compreende apenas os tempos relativos à execução de serviços Web e à transferências de dados. Já no cálculo dinâmico, também é considerada a carga de chamadas de serviços já atribuídas a cada sítios. O XCraft se baseia na comparação dinâmica de planos de materialização.

Em RUBERG *et al.* (2004a), é proposto um conjunto de funções para estimar os custos básicos de diferentes estratégias para a materialização de documentos AXML. Em especial, essas funções levam em conta operações típicas da invocação de serviços Web, tais como a serialização de dados XML e o empacotamento de mensagens SOAP. Vale ressaltar que esses custos são ignorados pelos otimizadores encontrados na literatura. A partir desses custos básicos, o XCraft usa um modelo analítico (descrito no Anexo I) para avaliar o desempenho de planos de materialização. O modelo proposto é bastante abrangente, pois considera sítios heterogêneos, diferentes enlaces de comunicação, execução paralela e a delegação de chamadas de serviços.

No XCraft, as funções de custos básicos de chamadas de serviços são combinadas para compor o custo de planos de materialização, conforme descrito no Anexo I. O modelo do XCraft trata dos custos de planos de materialização em três diferentes níveis de perspectiva. No primeiro nível do modelo, é estimado o custo de planos abstratos baseado na complexidade da otimização. Essa métrica ajuda o otimizador na escolha do método de geração de planos físicos equivalentes. Por exemplo, isso pode evitar que seja utilizado um método exaustivo em um plano abstrato cuja otimização iria demorar demais.

O segundo nível do modelo consiste em uma análise heurística para estimar o custo de planos parciais. Ele privilegia planos que envolvem um menor número de sítios próximos ao sítio mestre, visando reduzir os custos de comunicação. Essa análise

heurística pode ser interessante para filtrar planos e reduzir o espaço de busca, antes de uma análise completa. Já no terceiro nível, o modelo de custo usa as funções definidas RUBERG *et al.* (2004a) para calcular detalhadamente todos os custos envolvidos na materialização AXML. Vale ressaltar que uma dificuldade importante na otimização da materialização AXML decorre da necessidade de comparar planos inteiros, pois ao se tratar da delegação de chamadas de serviços, subplanos ótimos não garantem planos globais ótimos.

5.2.3 PRINCIPAIS TOPOLOGIAS DO PROBLEMA

A forma do grafo de dependências de um documento AXML exerce forte influência nas alternativas de avaliação em um sistema P2P, especialmente quanto às possibilidades de colaboração entre sítios. Ela também determina a ordem de avaliação das chamadas de serviços do documento.

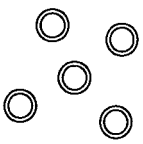
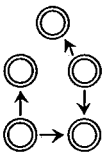
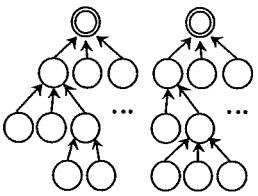
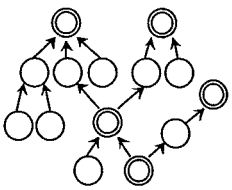
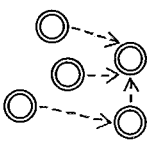
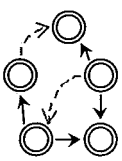
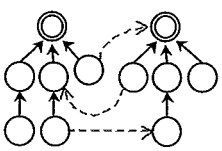
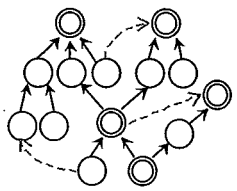
As principais topologias de grafos de dependências são classificadas na Tabela 4. Observe que um grafo é dito profundo se possui nodos cujos resultados não são persistentes no documento AXML. Basicamente, essa classificação considera os seguintes aspectos:

1) as seqüências de invocações provocadas por dependências entre chamadas de serviços envolvendo fluxos de dados temporários;
2) a presença de dependências compartilhadas no grafo, representados por nodos com $fanOut > 1$; e
3) o comprimento das seqüências de invocações decorrentes de chamadas colaterais.

A configuração mais simples de um grafo de dependências corresponde à topologia (a) da Tabela 4, que não envolve fluxos de dados e consiste em um conjunto de tarefas independentes. Em *grids*, esta topologia é bastante conhecida como "*bag of tasks*". A otimização deste tipo de grafo costuma tratar essencialmente do agendamento das invocações de serviços e do balanceamento de carga dos sítios. Em documentos AXML, além desses aspectos, o otimizador pode também reunir grupos de nodos e delegar a materialização desses grupos a outros sítios, favorecendo execuções paralelas. Aliás, em todas as topologias que representam grafos rasos, como nenhum nodo produz

resultados intermediários, não há possibilidade de redução dos custos de transferências de dados por meio da delegação de chamadas de serviços. Assim, a otimização enfoca sobretudo o agendamendo das invocações de serviços Web.

Tabela 4. Principais topologias de grafos de dependências.

	Grafo Raso		**Grafo Profundo**	
	fanOut ≤ 1	*fanOut* > 1	*fanOut* ≤ 1	*fanOut* > 1
sem chamadas colaterais	 (a)	 (b)	 (c)	 (d)
com chamadas colaterais	 (e)	 (f)	 (g)	 (h)

Em aplicações típicas de documentos AXML, como em ABITEBOUL, NGUYEN e RUBERG (2006), pode-se observar que grafos rasos não são freqüentes pois não contemplam o uso de parâmetros intencionais. Por outro lado, grafos profundos são mais comuns e permitem que o otimizador explore a proximidade entre sítios para delegar tarefas e reduzir o custo dos fluxos de dados. Uma configuração relevante desse tipo de grafo é representada pela topologia (c) da Tabela 4. Neste caso, os nodos de saída do grafo enraizam árvores que representam tarefas independentes. Como apenas as raízes dessas árvores têm resultados persistentes, a delegação de chamadas de serviços com resultados intermediários pode resultar em ganhos significativos de desempenho.

Vale observar que essa característica também ocorre na topologia (g) da Tabela 4, mesmo com a presença de chamadas colaterais. Já na topologia (d) da Tabela 4, o compartilhamento de dependências de invocação implica em pontos de sincronização

durante a materialização do grafo. Tais pontos indicam um acoplamento de dados entre chamadas de serviços, dificultanto a delegação de tarefas.

Por fim, a topologia (h) da Tabela 4 representa grafos de dependências cuja otimização é bastante complexa, pois há tanto compartilhamento de dependências como chamadas colaterais. No XCraft, para gerar planos de materialização, o primeiro passo da estratégia de otimização consiste em transformar grafos arbitrariamente complexos visando obter uma estrutura mais próxima da topologia (c), que oferece mais oportunidades de otimização.

5.3 ETAPAS DA ESTRATÉGIA DE OTIMIZAÇÃO

Para lidar com o dinamismo da materialização AXML, a idéia geral da estratégia de otimização desta tese consiste em dividir planos em partes menores e explorar a colaboração entre sítios para descentralizar a avaliação dessas partes. Inicialmente, o otimizador transforma o grafo de dependências em um conjunto de árvores, buscando obter uma representação semelhante à topologia (c) da Tabela 4. Assim, o otimizador pode começar a dividir o plano de acordo com as árvores geradoras do grafo de dependências. Cada árvore representa uma tarefa de materialização a ser tratada pelo otimizador.

Além disso, como um dos fatores de maior impacto no desempenho da materialização AXML é o custo dos fluxos de dados, o otimizador tenta quebrar o plano preservando alguns caminhos de dependências de invocação. Ou seja, ele divide cada tarefa de materialização em subplanos com uma altura menor. Assim, ele ainda consegue avaliar algumas dependências de invocação para explorar a delegação de chamadas de serviços, evitando transferências de resultados intermediários.

Vale mencionar que, embora na prática documentos XML não costumam ser muito profundos, é comum encontrar documentos com altura variando entre 6 e 9, tais como as bases de dados do *SwissProt*, do *eBay*, do *dblp* e da Nasa (MIGNET *et al.*, 2003). Em documentos AXML, espera-se que a altura dos planos de materialização seja semelhante a esses valores, ou até mesmo maior, devido ao uso de parâmetros intencionais não-concretos. Considerando a complexidade da busca por planos

eficientes, percebe-se que geralmente existe um grande número de alternativas de avaliação mesmo para documentos com altura entre 3 e 6. Este fato é evidenciado pelos resultados da análise do espaço de busca, discutida no Capítulo 6.

Os passos da estratégia proposta nesta tese são descritos na Figura 18 e correspodem ao algoritmo *DynamicOptimize* do Anexo I. Basicamente, o otimizador alterna etapas de planejamento e execução. Ele parte do grafo de dependências e de uma álgebra de operadores para gerar um plano abstrato inicial. Esse plano é dito abstrato porque o escopo de realização e o agendamento de invocações ainda não estão determinados.

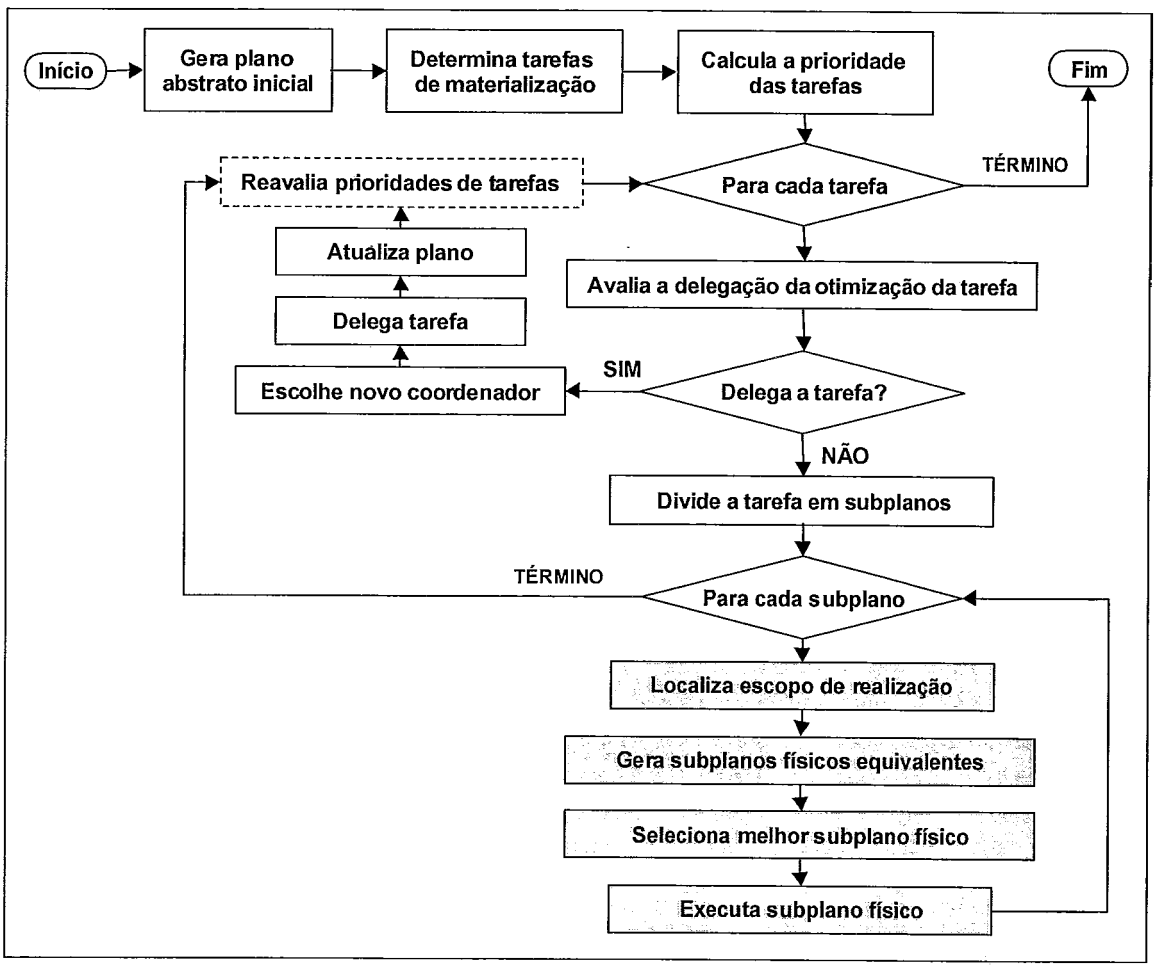


Figura 18. Passos da estratégia dinâmica e descentralizada para a otimização da materialização de documentos AXML.

O plano abstrato inicial é dividido em tarefas de materialização, tal que cada tarefa representa um resultado parcial do conteúdo do documento AXML. Então, o

otimizador atribui prioridades a essas tarefas e avalia se é conveniente delegar cada tarefa a algum sítio vizinho. Observe que a prioridade de uma tarefa pode ser determinada por diferentes métricas, tais como o número de nodos e o comprimento do caminho crítico. No XCraft, a otimização favorece a descentralização do processo de materialização AXML. Para isso, o otimizador atribui prioridades baseadas na análise do acoplamento de dados entre chamadas de serviços. Convém lembrar que as árvores do grafo podem ter dependências comuns, formando grupos de tarefas ligadas por um nodo compartilhado. Esses grupos se desfazem com a avaliação do nodo compartilhado, aumentando o potencial de paralelismo entre tarefas de materialização. No Anexo I, são apresentadas funções para calcular a prioridade das tarefas de materialização baseada neste potencial

Para cada tarefa a ser avaliada localmente, o otimizador explora a ordem topológica dos nodos para quebrá-la em subplanos de altura k. O parâmetro k é chamado de fator de quebra da otimização dinâmica e pode ser fixo ou estimado pelo otimizador. Por exemplo, o otimizador pode sempre dividir as tarefas de materialização em partes iguais, baseado na altura de cada tarefa. Contudo, geralmente a altura dos subplanos contribui bastante para aumentar o tamanho do espaço de planos equivalentes. Portanto, o otimizador deve sempre combinar a escolha do fator de quebra com um método adequado para a geração de alternativas de avaliação.

A análise de cada subplano é o ponto central da otimização dinâmica. Para cada subplano de materialização, o XCraft identifica o escopo de realização (*i.e.*, os provedores dos serviços Web e os possíveis controladores) e enumera subplanos físicos equivalentes. Vale ressaltar que no XCraft a localização de serviços Web é postergada para o momento de geração dos subplanos. Isso contribui para a adaptação às mudanças no sistema P2P. A seleção do melhor subplano físico é feita pelo tempo de materialização, estimado com o modelo de custo do XCraft. Por fim, o melhor subplano físico é executado.

Esse processo é repetido até que todos os subplanos de todas as as tarefas de materialização sejam concluídas. Observe que opcionalmente o otimizador pode ainda recalcular as prioridades das tarefas após cada avaliação, para levar em conta as eventuais mudanças de potencial de paralelismo, decorrentes da materialização de

dependências compartilhadas. A seguir, são apresentas técnicas para produzir planos iniciais de materialização a partir de grafos de dependências.

5.4 GERAÇÃO DE PLANOS ABSTRATOS

No XCraft, a geração de planos de materialização é incremental. Ou seja, inicialmente o otimizador produz planos incompletos que são sucessivamente refinados até que os operadores sejam físicos (*i.e.*, possam ser executados). Essa geração é baseada na transformação do grafo de dependências em uma estrutura arborescente e utiliza uma álgebra de operadores para codificar as alternativas de materialização AXML.

5.4.1 ÁRVORES GERADORAS

O grafo de dependências de um documento AXML pode ser arbitrariamente complexo, contendo dependências compartilhadas e chamadas colaterais. Tal estrutura dificulta tanto a geração incremental de planos de materialização como a colaboração P2P. Conseqüentemente, o otimizador precisa aplicar algumas transformações no grafo para facilitar a geração de planos de materialização.

Baseado nos pontos de saída de um grafo de dependências, o XCraft constrói uma floresta de árvores geradoras, a qual é utilizada para produzir um plano inicial de materialização. Os nodos cujos resultados são persistentes no documento AXML enraizam as árvores desta floresta. Isto é, uma árvore do plano representa claramente uma parte do documento AXML a ser materializada. Além disso, a floresta gerada pelo XCraft ignora a ordem das árvores e é mínima, conforme descrito no Anexo I. Esta floresta é chamada de MFST (de "*Minimum Forest of Spanning Trees*") do grafo de dependências.

Há algoritmos clássicos para construir as árvores geradoras de um grafo. Um exemplo bastante popular é o algoritmo de *Prim* (AHO *et al.*, 1983), que tem complexidade de tempo da ordem de $O(|V|^2)$ para grafos representados por listas de adjacência e $O(|V|.log|V| + |E|)$ para *heaps*. Nesses algoritmos, os nodos do grafo são percorridos um a um para formar as árvores geradoras, baseados na vizinhança entre

eles. Esse percurso segue a direção das arestas. Para documentos AXML, algumas alterações são necessárias. Primeiro, as sementes do algoritmo (*i.e.*, os nodos usados para começar os percursos) são limitadas aos nodos de saída do grafo de dependências. Além disso, o percurso ocorre no sentido inverso das arestas colaterais e cada chamada colateral é representada com uma nova árvore da MFST. As arestas colaterais do grafo de dependências são representadas como anotações nos nodos do plano de materialização, criando relacionamentos entre as árvores geradoras.

Se todos os subgrafos de um grafo de dependências são disjuntos, então o otimizador pode identificar facilmente as partes independentes do plano de materialização. Porém, grafos contendo dependências compartilhadas e chamadas colaterais não têm uma correspondência natural com árvores. Neste caso, são necessários alguns artifícios para construir a MFST, forçando que cada nodo pertença a somente uma árvore. Nesta tese, são propostas as seguintes operações de transformação:

- a replicação de nodos, que substitui uma chamada de serviço do grafo por um conjunto de cópias que representam uma única invocação; e
- a clonagem de nodos, que consiste em acrescentar novas instâncias de uma chamada de serviço no grafo de dependências.

Essas operações são baseadas no destacamento de nodos compartilhados e colaterais do grafo de dependências. Elas visam obter uma forma arborescente do grafo de dependências, em uma estrutura mais adequada para a avaliação distribuída. A operação de replicação de nodos é usada para separar subgrafos conectados por dependências compartilhadas, enquanto a clonagem serve para representar múltiplas invocações decorrentes de chamadas colaterais. Vale ressaltar que essas transformações preservam todas as restrições de invocação entre as chamadas de serviços do grafo.

No Anexo I é descrito um algoritmo para expandir grafos de dependências pela replicação e clonagem de todas as dependências compartilhadas e chamadas colaterais. A partir do grafo expandido, o otimizador gera a MFST correspondente.

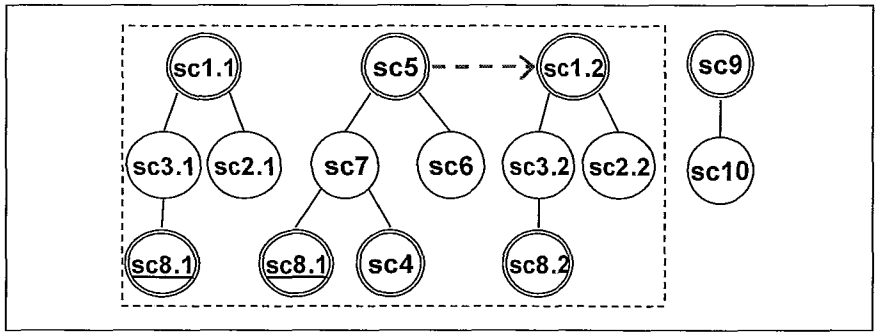


Figura 19. Árvores geradoras do documento *SwapWorkspace*.

A Figura 19 mostra a MFST obtida do grafo de dependências do documento *SwapWorkspace*. Anotações colaterais são denotadas por setas tracejadas e as chamadas colaterais possuem um sufixo para indicar invocações distintas. Os nodos replicados têm a identificação sublinhada e círculos duplos representam nodos persistentes. Note que as árvores geradoras de um grafo de dependências podem ser reunidas em grupos (possivelmente com alguma interseção entre os grupos). Cada grupo envolve todas as árvores que contêm réplicas de um nodo. Por exemplo, o grupo do nodo compartilhado sc8 é indicado pelo retângulo tracejado na Figura 19. As árvores de um grupo estão relacionadas entre si e não podem ser avaliadas independentemente umas das outras, devido ao acoplamento de dados entre elas.

5.4.2 Álgebra de Operadores para Materialização AXML

Um dos pilares da estratégia de otimização do XCraft consiste em uma álgebra de operadores para codificar planos de materialização levando em conta uma avaliação incremental e a colaboração entre sítios. Esses operadores tratam de vários aspectos da otimização das chamadas de serviços de um documento AXML, tais como a localização do escopo de realização e a atualização do plano com respostas ativas.

Os operadores da álgebra do XCraft são descritos na Tabela 5. Maiores detalhes podem ser encontrados no Anexo I. Basicamente, esses operadores podem ser agrupados em três tipos:

- operadores abstratos μ e ρ, que representam as possíveis combinações de escopos de execução e de delegação de uma chamada de serviço;

- operadores físicos **invoke**, **fetch** e δ, que descrevem todas as informações necessárias para a invocação de um serviço Web; e
- operadores auxiliares θ, **locate** e **pipe**, que visam apoiar o processo de otimização descentralizada.

Tabela 5. Álgebra de operadores para a otimização dinâmica e descentralizada da materialização de documentos AXML.

OPERADOR	DESCRIÇÃO
$\mu(v)$	O operador *materialize* indica a geração de um plano de invocação para a chamada de serviço v, considerando as dependências e chamadas colaterais de v.
$\rho(v)$	O operador *retrieve* trata da geração de um plano de invocação para um nodo replicado v. Ele funciona como $\mu(v)$, porém com um acesso prévio ao *cache* do otimizador, garantindo que v seja invocado somente uma vez.
invoke(v, p_v)	Representa a invocação da chamada de serviço v, usando o executor e o coordenador estabelecidos no plano de invocação p_v= <*Pe*,*Pc*>.
fetch(v)	Indica a recuperação do resultado do nodo replicado v, a partir do *cache* do otimizador ou pela invocação de v, caso o resultado de v ainda não tenha sido materializado.
$\delta(v, p_v)$	O operador *delegate* denota a delegação do controle de execução do subplano enraizado em v para o sítio *Pc* de p_v= <*Pe*,*Pc*>. Ou seja, *Pc* deverá invocar v em *Pe* e retornar o resultado para o sítio mestre.
$\theta(v)$	O operador *optimize* permite que o sítio mestre solicite a algum vizinho a geração de um plano físico para o subplano enraizado em v.
locate(v)	Representa uma solicitação para que um sítio vizinho localize o escopo de realização do subplano enraizado em v.
pipe	Usado para simplificar a atualização do plano de materialização com respostas ativas. Este operador retorna a união de seus parâmetros de entrada.

Esses operadores formam a álgebra **A**, usada na definição de planos de materialização. Observe que tanto os operadores físicos como os auxiliares representam invocações concretas de serviços Web. Porém, análogo aos operadores auxiliares, o operador de delegação de execução (*i.e.*, δ) aponta para um serviço Web de colaboração P2P. No caso da execução de operadores auxiliares, o otimizador deve escolher um sítio vizinho para executar o operador. Devido ao dinamismo do sistema, tal escolha é feita preferencialmente no momento anterior à execução do operador.

Os operadores ρ e **fetch** visam tratar o compartilhamento de resultados de chamadas de serviços. Eles pressupõem que otimizador mantém um *cache* para armazenar os resultados das primeiras solicitações relativas a dependências compartilhadas. Assim, as demais solicitações recuperam o resultado do *cache*, em vez de disparar a chamada de serviço. Esse *cache* pode registrar tanto o plano de invocação escolhido para a chamada de serviço (a ser recuperado pelo operador ρ) como o resultado da invocação (para o operador **fetch**).

O otimizador XCraft usa os operadores algébricos de para refinar sucessivamente um plano de materialização. Ao serem avaliados pelo otimizador, os operadores abstratos são transformados em operadores físicos, conforme a escolha dos respectivos planos de invocação. Em particular, o operador μ pode ser transformado em um operador **invoke** ou em δ. As transformações do plano também podem ocorrer pela substituição de operadores. Por exemplo, um operador μ pode ser substituído pelo operador **locate** caso o sítio mestre não consiga localizar o escopo de execução de uma chamada de serviço. As principais regras de transformação algébrica do plano são apresentadas no Anexo I.

Vale ressaltar que a álgebra proposta é extensível e permite que tanto novos operadores como novas regras de transformação sejam acrescentados ao XCraft.

5.4.3 Plano Abstrato Inicial

A partir da MFST de um grafo de dependências, o otimizador gera um plano inicial baseado na álgebra de operadores proposta. Idealmente, o tempo gasto nesta etapa da otimização deve ser irrisório. Para isso, o XCraft explora os operadores

abstratos da álgebra, que ajudam a simplificar sobremaneira a geração do plano inicial pois permitem a localização postergada de serviços Web.

O Anexo I apresenta um algoritmo para a geração de planos iniciais. Basicamente, para cada nodo da MFST de um grafo de dependências, é criado um operador abstrato no plano de materialização, tal que nodos replicados são rotulados com o operador ρ. Oportunamente, o otimizador pode rotular um nodo com o operador θ para indicar que o respectivo subplano não será avaliado localmente. Esse mecanismo de delegação da otimização apresenta várias vantagens, sendo especialmente útil para lidar com problemas de sobrecarga do sítio mestre.

A Figura 20 mostra o plano abstrato inicial gerado a partir do grafo de dependências do documento *SwapWorkspace*, conforme o algoritmo proposto no Anexo I. Cada operador do plano representa uma chamada de serviço do documento AXML. As chamadas colaterais são indicadas por anotações do tipo "*cp*" nos operadores, quando for o caso. Por exemplo, na Figura 20, o operador μ(sc5) aponta para a chamada colateral que é representada pelo operador μ(sc1.2). Outras informações também podem ser anotadas nos operadores, para auxiliar a otimização. No exemplo do documento *SwapWorkspace*, os nodos estão anotados com a altura do operador no plano.

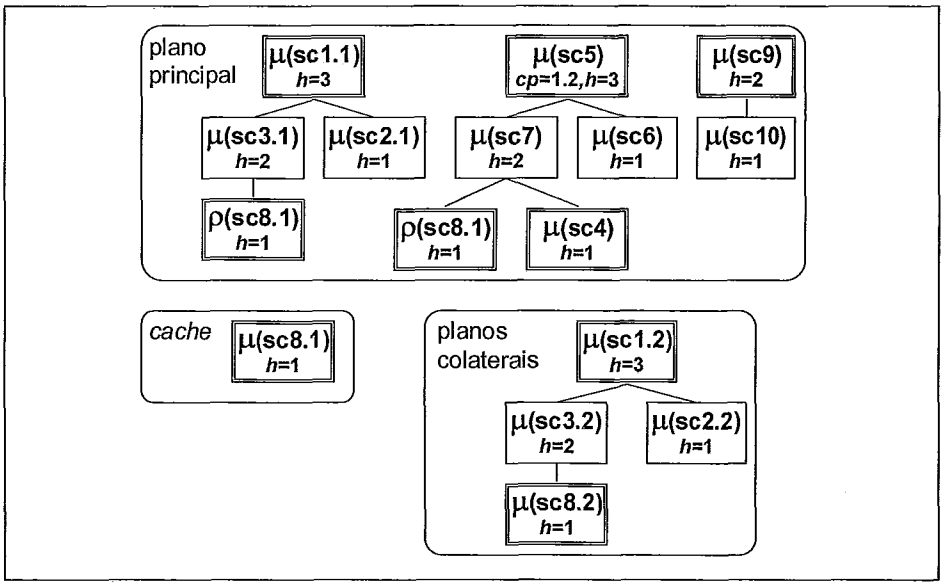


Figura 20. Plano abstrato inicial gerado para o documento *SwapWorkspace*.

No XCraft, para reduzir os requisitos do armazenamento de um plano de materialização, as sub-árvores enraizadas por nodos replicados são destacadas e

substituídas por operadores ρ ou **fetch**. Isso pode ser observado no exemplo da Figura 20. Deste modo, o plano de materialização de um documento AXML fica organizado em três áreas:

- o plano principal reúne os operadores que representam as árvores geradoras do grafo de dependências, tal que operadores ρ e **fetch** denotam subplanos dos nodos replicados;
- o cache de operadores replicados contém as sub-árvores dos nodos replicados, que são relativos a dependências compartilhadas; e
- a área de planos colaterais abrange as sub-árvores das chamadas colaterais do documento AXML.

Caso não haja respostas ativas na materialização de um documento AXML, o otimizador pode armazenar o plano inicial de um documento AXML visando agilizar solicitações futuras.

5.5 GERAÇÃO DE SUBPLANOS FÍSICOS

As tarefas de materialização de um documento AXML geralmente não são uma boa unidade para dividir um plano abstrato e gerar subplanos físicos, pois individualmente elas ainda representam enormes espaços de alternativas de materialização. Por isso, o XCraft quebra essas tarefas de acordo com a ordem topológica dos operadores. Isto é, o otimizador usa a altura dos operadores para gerar subplanos. Essa forma de divisão permite analisar fluxos de dados entre chamadas de serviços do documento AXML e aplicar convenientemente a delegação da execução.

5.5.1 IDENTIFICAÇÃO DE PONTOS DE QUEBRA

Para dividir uma tarefa de materialização, o otimizador usa um parâmetro chamado fator de quebra. Assim, os pontos de quebra de uma tarefa são os nodos cujas sub-árvores tem a menor altura igual ou inferior ao fator de quebra. Por definição, o operador raiz de uma tarefa de materialização também é considerado um ponto de

quebra. O algoritmo *SplitTask* do Anexo I é usado pelo XCraft para identificar as raízes dos subplanos de um tarefa de materialização.

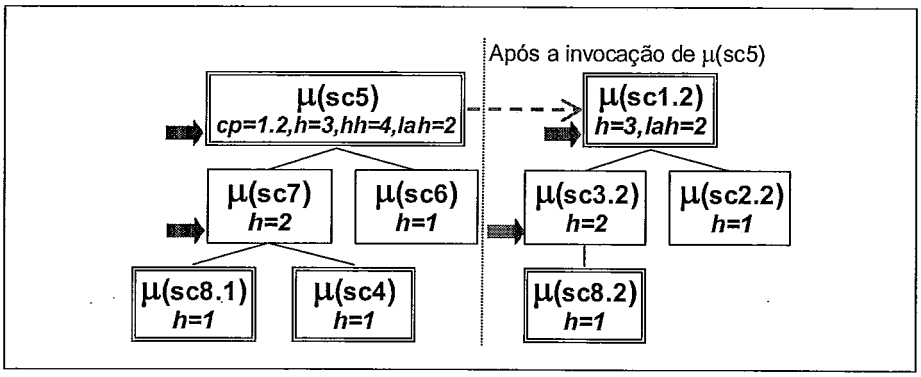


Figura 21. Pontos de quebra da tarefa de materialização.

A Figura 21 mostra os subplanos da tarefa de materialização que enraizada pelo operador μ(sc5) no documento *SwapWorkspace*. Os pontos de quebra estão representados por setas largas e a menor altura dos operadores é indicada por anotações "lah" nos nodos. Neste exemplo, pressupõe-se que o otimizador adota um fator de quebra k=2.

Embora no exemplo da Figura 21 também esteja incluído o subplano da chamada colateral μ(sc1.2), observe que ele só é considerado pelo otimizador para calcular os pontos de quebra após a invocação de μ(sc5). Isso ocorre porque o otimizador pressupõe que uma chamada colateral e todas as suas dependências somente são executadas após o nodo que a dispara. Contudo, essa premissa pode ser relaxada pelo usuário. Neste caso, o otimizador deve usar a menor altura absoluta dos nodos para achar os pontos de quebra da tarefa de materialização. A altura absoluta leva em conta tanto arestas simples como colaterais.

5.5.2 AGENDAMENTO DA AVALIAÇÃO DE SUBPLANOS

Basicamente, os subplanos de uma tarefa de materialização são analisados em ordem topológica ascendente (*i.e.*, a partir das folhas), tal que os nodos filhos de um operador são avaliados primeiro. Entretanto, existem subplanos que são independentes entre si e o otimizador pode ordená-los em várias seqüências diferentes. Logo, o otimizador deve determinar uma ordem de avaliação. Isso significa que o otimizador começa a definir o agendamento das invocações da tarefa de materialização. Contudo,

vale ressaltar que esse agendamento não precisa ser feito para todos os operadores da tarefa, mas somente para os pontos de quebra.

Para controlar a ordem de avaliação dos subplanos de uma tarefa, o otimizador se baseia em uma estrutura auxiliar chamada de guia-S. Essa estrutura contém os pontos de quebra da tarefa e representa os relacionamentos de dependências entre subplanos. Observe que dois subplanos podem ser nodos irmãos no guia-S mesmo se os respectivos pontos de quebra não são irmãos na tarefa de materialização.

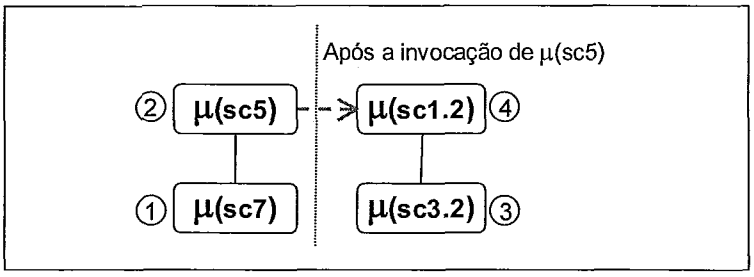


Figura 22. Guia-S para ordenar a avaliação dos subplanos da tarefa de μ(sc5).

A mostra o guia-S da tarefa de materialização enraizada em μ(sc5). Círculos numerados indicam a ordem de avaliação dos subplanos. Note que neste exemplo não há subplanos concorrentes e basta que o otimizador siga a ordem topológica dos nodos.

Em tarefas com subplanos concorrentes, o otimizador pode usar diferentes heurísticas para determinar a ordem de avaliação dos subplanos do guia-S. Em geral, essas heurísticas são baseadas ou na ordenação dos filhos de cada nodo do guia-S, ou em um lista de pontos de quebra prontos para avaliação (*i.e.*, com todas as dependências devidamente resolvidas). Na primeira abordagem, é usada alguma política para atribuir prioridades ao filhos de cada ponto de quebra. Por exemplo, o otimizador pode determinar que subplanos com maior caminho crítico são avaliados primeiro.

Na segunda abordagem para ordenar a avaliação dos subplanos de uma tarefa, o agendamento é feito a partir de uma lista de nodos prontos. Inicialmente, essa lista contém todos os nodos-folhas do guia-S. Então, o otimizador aplica alguma política de atribuição de prioridades para ordenar apenas os nodos dessa lista. Assim, a heurística de agendamento considera todos os subplanos da tarefa, em vez de ser localizada nos filhos de cada operador.

5.5.3 Enumeração das Alternativas de Materialização

Esta etapa consiste no núcleo do processo de otimização do XCraft, na qual são produzidos os subplanos físicos que controlam a materialização de um documento AXML. Observe que um subplano abstrato representa um "*template*" que o XCraft usa para gerar subplanos físicos equivalentes. Assim, o otimizador começa a geração de subplanos físicos pela localização do escopo de realização do subplano abstrato. Ele percorre o subplano abstrato anotando os escopos de execução e de delegação de cada operador. Caso o sítio mestre não disponha dessas informações, o operador abstrato é substituído por um operador **locate**. Ao final da anotação do escopo de realização, o otimizador verifica se operadores **locate** podem ser agrupados, visando reduzir os custos de comunicação na localização dos serviços Web.

Se a anotação do escopo de realização de um subplano abstrato resulta em muitos operadores **locate**, então o otimizador pode decidir delegar a otimização do subplano pela substituição da raiz do subplano por um operador θ.

A partir do escopo de realização, o otimizador enumera as alternativas de materialização AXML e seleciona o melhor subplano físico. Convém lembrar que, além da seleção de sítios, o otimizador deve determinar também o agendamento das invocações do subplano. Porém, nesta tese, o enfoque da otimização é na seleção de sítios, visando reduzir principalmente custos de transferências de dados. Ou seja, o otimizador primeiro seleciona os sítios para, então, agendar a avaliação dos operadores. Em linhas gerais, o otimizador realiza o seguinte:

1) gera o espaço de busca formado por subplanos parciais, considerando apenas o escopo de execução dos operadores;

2) para cada subplano parcial, gera o espaço de busca formado por subplanos físicos, a partir do escopo de delegação dos operadores; e

3) para cada subplano físico, gera as alternativas de agendamento de invocações.

Para enriquecer essa abordagem, o XCraft se baseia em um modelo de custo que leva em conta a carga de chamadas de serviços atribuídas a cada sítio envolvido em um subplano. Além disso, o otimizador pode usar diferentes heurísticas para realizar esses

passos, conforme o método de geração adotado no subplano. Em um método exaustivo, todas as combinações possíveis são avaliadas e um subplano físico ótimo é selecionado. Entretanto, devido ao grande número de alternativas, métodos exaustivos são geralmente inviáveis. Assim, o otimizador pode explorar heurísticas para reduzir o número de subplanos considerados em cada passo. Por exemplo, em um método guloso simples, a seleção de sítios é baseada apenas no escopo de execução dos sítios, ignorando as possibilidades de delegação (passo 2).

Uma outra abordagem de geração de subplanos físicos consiste em usar uma heurística de contexto (RUBERG *et al.*, 2004a) para restringir os possíveis colaboradores para delegação dos operadores, considerando apenas o conjunto de possíveis executores. No Anexo I são discutidos os principais métodos de geração de subplanos físicos. Além disso, em PEREIRA, RUBERG e MATTOSO (2006) é proposto um método baseado em busca local estocástica para a geração de planos de materialização. Esse método permite estabelecer limites para o tempo gasto na otimização do plano, tendo mostrado bons resultados mesmo com diferentes heurísticas de agendamento de invocações.

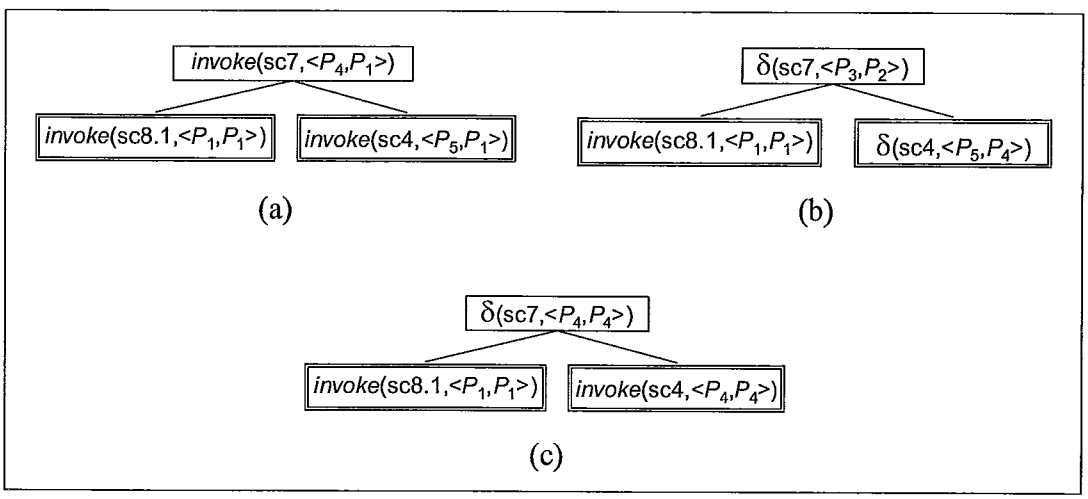


Figura 23. Alguns subplanos físicos equivalentes gerados para o operador μ(sc7).

A Figura 23 ilustra algumas alternativas de materialização para o subplano do operador μ(sc7). Vale destacar que a estratégia proposta nesta tese beneficia o uso de métodos de busca exaustiva, geralmente inviáveis na materialização AXML, pois o otimizador lida com planos de tamanho reduzido. Além disso, o XCraft considera diferentes políticas de agendamento de invocações.

5.5.4 EXECUÇÃO DE SUBPLANOS FÍSICOS

Cada subplano físico selecionado pelo otimizador é avaliado em uma abordagem ascendente (*i.e.*, das folhas para a raiz), seguindo o agendamento de invocações estabelecido no subplano. Porém, como a materialização AXML é descentralizada no XCraft, alguns operadores do plano podem representar delegações de chamadas de serviços. Ou seja, existem sub-árvores cujos operadores serão avaliados remotamente. Conseqüentemente, o otimizador precisa avaliar apenas os operadores locais e os pontos de delegação. Os operadores locais de plano são aqueles cujo controlador é o sítio mestre, enquanto os pontos de delegação são nodos rotulados com operadores δ ou θ. Convém lembrar que um plano é dito físico se seu escopo de realização está totalmente definido. Ou seja, se todos os operadores locais do plano são do tipo físico ou auxiliar. Além disso, pressupõe-se que os resultados de pontos de delegação são sempre retornados ao sítio mestre.

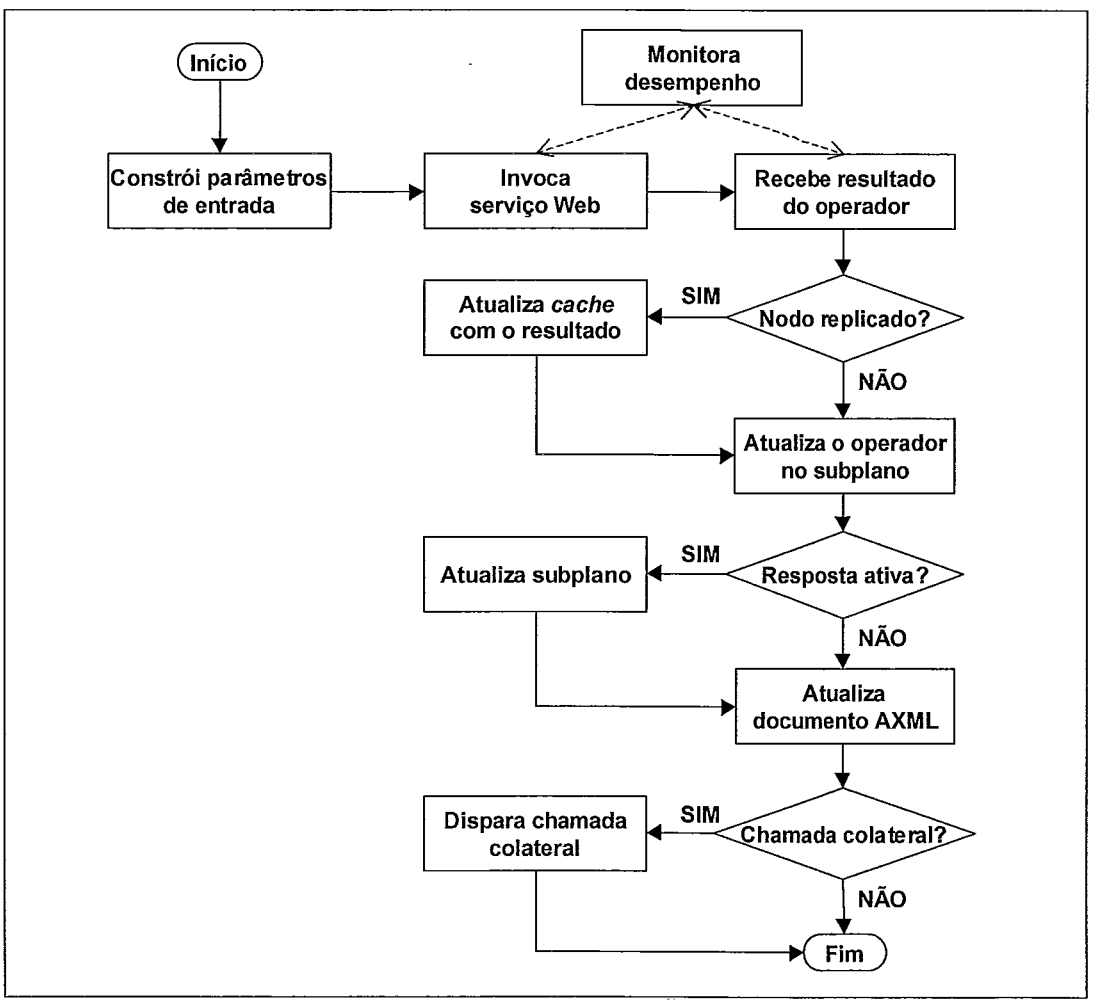


Figura 24. Passos da avaliação de cada operador local de um subplano físico.

A Figura 24 mostra os principais passos da avaliação de cada operador de um subplano físico. O primeiro passo consiste em construir as entradas do operador, que podem ser: ou dados AXML que alimentam chamadas de serviços do documento, tratadas por operadores **invoke** ou **fetch**; ou planos de materialização remota, que alimentam chamadas de serviços para colaboração P2P. Em seguida, o otimizador invoca o serviço Web do operador e recebe o resultado. O desempenho desses passos é monitorado para que o otimizador possa detectar previamente problemas com longas esperas e falhas de execução. Eventualmente, se o desempenho de um operador for muito aquém do esperado, o otimizador pode decidir regerar partes do subplano físico.

Após receber o resultado da avaliação de um operador, o otimizador realiza uma série de verificações. Caso o operador envolva um nodo replicado, o *cache* de planos do otimizador é atualizado com o resultado. Note que nodos replicados podem estar contidos no plano de materialização remota de um operador de colaboração P2P. Continuando, o otimizador atualiza o estado no operador no subplano e verifica se o resultado da avaliação contém respostas ativas. Se for o caso, o otimizador usa operadores **pipe** para acrescentar as novas chamadas de serviços ao subplano. Vale ressaltar que a atualização do subplano com respostas ativas ocorre preferencialmente antes da atualização do documento AXML, porque ela permite verificar a validade da atualização. Por fim, a chamada colateral do operador é devidamente disparada.

5.6 COMENTÁRIOS

Além de quebrar tarefas de materialização verticalmente, baseado na altura dos operadores no plano, o XCraft também pode agrupar nodos horizontalmente, por intermédio dos operadores **pipe** e θ. Esta técnica é bastante interessante para tratar grafos de dependências do tipo "*bag of tasks*" e grafos cujos nodos têm o *fanIn* elevado (*i.e.*, muitos parâmetros intencionais).

Um diferencial importante do otimizador XCraft consiste no apoio à descentralização da materialização AXML. Na estratégia proposta, a descentralização é explorada em vários aspectos, tais como a localização de provedores de serviços Web, a geração de planos de materialização e a delegação do controle de execução. Esses pontos são tratados explicitamente na álgebra de operadores de materialização AXML,

permitindo que o otimizador tenha maior flexibilidade de adaptação ao dinamismo de sistemas P2P. Outro benefício relevante da estratégia proposta é que respostas ativas afetam apenas as decisões do subplano que está sendo avaliado pelo otimizador. Deste modo, são evitadas re-otimizações desnecessárias.

Como álgebra do XCraft é baseada em serviços Web, os planos de materialização remota podem ser serializados no próprio formato de documentos AXML. Além disso, informações sobre o andamento da materialização de um documento AXML podem ser enviadas no cabeçalho de plano remotos. Por exemplo, para prevenir que planos sejam repassados indefinidamente na delegação da otimização, o XCraft inclui no cabeçalho a trilha de sítios pelos quais o plano de materialização passou. O plano também pode conter o número máximo de delegações permitidas.

Capítulo 6

ANÁLISE DAS TÉCNICAS PROPOSTAS

Neste capítulo, as técnicas de otimização do XCraft são avaliadas por meio de prototipação e de resultados empíricos. Esses experimentos foram realizados em diferentes cenários e com diversas topologias de documentos AXML, baseadas em padrões típicos de controle de fluxo e de dados em workflows. Em todos os cenários, foram observados ganhos importantes de desempenho da estratégia do XCraft.

6.1 INTRODUÇÃO

Neste capítulo, as técnicas de otimização propostas são avaliadas por meio de resultados empíricos obtidos com um protótipo do otimizador XCraft. As configurações e artefatos utilizados nesses testes são descritos na seção 6.2, abrangendo o protótipo XCraft, a rede P2P, documentos AXML e serviços Web.

Resultados obtidos por simulação são apresentados na seção 6.3, em uma avaliação do crescimento do espaço de planos alternativos na materialização de documentos AXML. Essa análise considera fatores relevantes como o número de sítios participantes e o aninhamento de dependências de invocação. São comparados diferentes métodos de geração de planos de materialização, com destaque para o efeito da estratégia proposta no XCraft. Na seção 6.4, utiliza-se o tempo de materialização de documentos AXML para avaliar o desempenho das estratégias centralizada e com delegação de tarefas. Esses resultados mostram ganhos significativos de desempenho com o XCraft. A seção 6.5 aborda resultados obtidos com o XCraft por meio de um método estocástico para a geração de planos de materialização. Por fim, a seção 6.6 trata de investigações sobre técnicas de otimização complementares ao XCraft, a partir da proposta de um modelo de fragmentação para bases de documentos XML.

6.2 AMBIENTE DE TESTE

Nesta tese, a análise de resultados experimentais abrange dois tipos de testes: reais, nos quais os documentos AXML são efetivamente materializados pela execução dos serviços Web necessários; e simulados, cujas execuções de serviços Web são calculadas ou substituídas por tempos de espera pré-determinados. Vale mencionar que, exceto em análises com grandes espaços de busca, os passos de otimização são sempre executados no XCraft, tanto em testes reais com em simulados. Nesta seção, são descritas todas as configurações e recursos utilizados nos testes realizados.

6.2.1 PROTÓTIPO DO XCRAFT

Foi construído um protótipo do XCraft na linguagem de programação Java (com a JDK 1.4.2), sobre a plataforma *ActiveXML* na versão 4-Beta (ACTIVEXML, 2002). Esse protótipo é multi-tarefa e todo baseado em código livre, utilizando o servidor Web *Apache Tomcat* 4.1.29 (TOMCAT, 1999) como infra-estrutura básica. Também foram usadas as bibliotecas: *Apache Axis* 1.1 (AXIS, 2000), que implementa o protocolo SOAP; e JDSL (de "*Java Data Structure Library*") versão 2.1 (JDSL, 2000), para manipulação de grafos.

A execução de testes em sistemas P2P geralmente envolve um esforço considerável, pois requer a disponibilidade e a configuração de um conjunto de sítios autônomos. Para permitir uma análise abrangente do desempenho da materialização de documentos AXML, o protótipo do XCraft foi encapsulado no ambiente de simulação SiMAX (PEREIRA, RUBERG e MATTOSO, 2006). A partir de arquivos de configuração que descrevem tanto as chamadas de serviços de um documento AXML como os recursos disponíveis no sistema P2P, o SiMAX permite disparar o otimizador XCraft para gerar e avaliar planos de materialização. Assim, é possível observar o comportamento do otimizador em diversos cenários.

6.2.2 REDE PONTO-A-PONTO

Nos testes reais, foi utilizada uma rede P2P contendo três sítios interconectados conforme a Figura 25. Os sítios estão distribuídos em duas sub-redes conectadas pela Internet. A comunicação na Internet ocorre através de enlaces de 512Kbps e a rede local está interconectada a 36Mbps. Observe que essa configuração permite explorar situações relevantes para a materialização de documentos AXML, especialmente quando há possibilidade de delegação de tarefas.

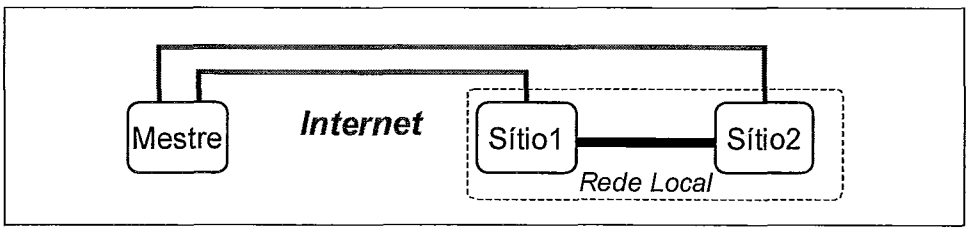


Figura 25. Sistema P2P utilizado nos testes experimentais reais.

Tabela 6. Configuração dos sítios usados em testes experimentais reais.

Sítio	S.O.	*BogoMips*	RAM (Mb)
Mestre	*Linux Debian*	2957,31	512
Sítio1	*MS Windows XP*	1718,18	512
Sítio2	*Linux SuSe*	1198,77	512

Na Tabela 6 são detalhadas as características das máquinas usadas nos testes reais. São indicados o sistema operacional, o desempenho da CPU em *BogoMips* (DORST, 2006) e a memória RAM instalada na máquina (em *Mega bytes*). Vale ressaltar que, apesar de possuir um número pequeno de sítios, o sistema P2P usado nos testes abrange máquinas com diferentes cargas de processamento e enlaces de comunicação.

6.2.3 GERAÇÃO DE DOCUMENTOS AXML

A base de documentos AXML utilizados nos testes foi construída visando permitir analisar diferentes topologias de grafos de dependências. Para facilitar essa tarefa, foi desenvolvido um gerador AXML para produzir documentos com diferentes serviços Web, comprimentos de caminho crítico (relativos às alturas de aninhamento de dependências de invocação) e número de tarefas independentes.

Foram geradas configurações de padrões de fluxos de controle tipicamente encontrados em ambientes para execução de *workflows* distribuídos (AALST *et al.*, 2003). Essas configurações são baseadas em:

(i) tarefas independentes;

(ii) tarefas com seqüências de invocações; e

(iii) tarefas com sincronização.

Essas configurações são bem comuns em *workflows* científicos e com serviços Web. Em particular, *workflows* do tipo (i) são conhecidos como "*bag of tasks*" (VARGAS *et al.*, 2004), enquanto os do tipo (*ii*) se aproximam de cargas de processamento em *grids* chamadas de "*batch pipelined*" (THAIN *et al.*, 2003). Já em aplicações ligadas à astronomia, geralmente são encontrados *workflows* semelhantes ao

tipo (*iii*), como mostrado em FOSTER *et al.* (2002) e em BLYTHE *et al.* (2005). A Figura 26 ilustra a configuração dos grafos de dependências correspondentes a essas configurações de documentos AXML.

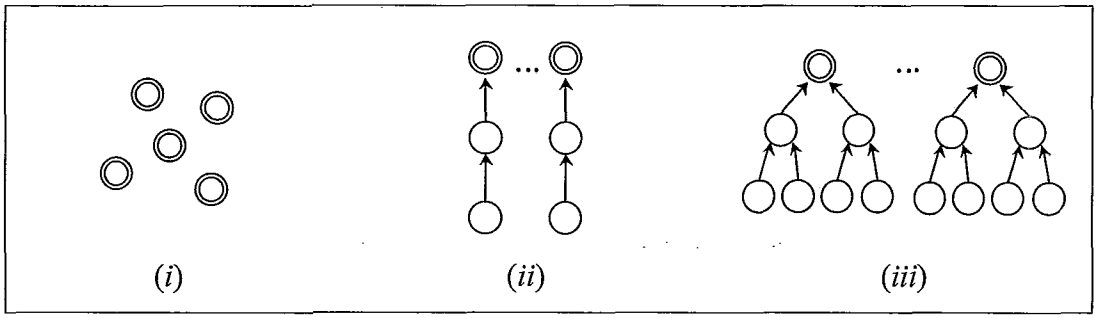


Figura 26. Tipos de grafos de dependências dos testes experimentais.

6.2.4 SERVIÇOS WEB

Os testes foram realizados na plataforma *ActiveXML*, explorando recursos para publicação de serviços Web declarativos. Convém lembrar que esses serviços são descritos por consultas sobre documentos do repositório local. Vale ressaltar também que esses serviços poderiam ser substituídos por serviços Web regulares sem afetar o desempenho das técnicas de otimização avaliadas.

```
<?xml version="1.0" encoding="UTF-8"?>
<serviceDefinition type="query" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
axml:docName="Service50k" xmlns:axml="http://www-rocq.inria.fr/verso/AXML">
.....<parameters>
..... ....<param name="_documentIn"/>
.....</parameters>
.....<definition>
.....<query> <![CDATA[
..... ....select e1 from e1 in XMark50k/site;
.....]]> </query>
.....</definition>
</serviceDefinition>
```

nome do serviço Web declarativo

parâmetro de entrada

consulta X-OQL

Figura 27. Serviço Web declarativo que retorna um documento com 50K*bytes*.

Foram criados serviços declarativos que realizam consultas sobre documentos armazenados no repositório local do sítio *ActiveXML*. Esses arquivos locais são baseados em documentos XML do *benchmark* XMARK (XMARK, 2003), gerados com a ferramenta ToxGene (BARBOSA *et al.*, 2002). O tamanho dos documentos XML foi variado e cada serviço declarativo acessa um determinado documento, visando obter resultados de invocações com diferentes volumes de dados. Por exemplo, a Figura 27

mostra a especificação do serviço declarativo *Service50k*, de acordo com a linguagem utilizada no sistema *ActiveXML* (ACTIVEXML TEAM, 2005). Este serviço retorna o conteúdo do documento "XMark50k", que possui aproximadamente 50*Kbytes*.

Observe que os serviços declarativos gerados permitem controlar o fator de seleção e de expansão, respectivamente, dos resultados das chamadas de serviços em relação às suas entradas.

6.3 ANÁLISE DO ESPAÇO DE BUSCA

Por ser um problema de otimização combinatória, a materialização eficiente de documentos AXML costuma envolver a geração de um grande número de planos alternativos. Isso ocorre porque o percurso do espaço de busca deve ser guiado por métricas que levam em conta diferentes fatores de desempenho, especialmente os efeitos da delegação de chamadas de serviços. Para melhor compreender os aspectos que determinam o tamanho desse espaço de busca, foram feitas simulações com diferentes cenários, variando:

- o número de sítios envolvidos na materialização, que reflete o tamanho do sistema P2P;
- o *fanIn* dos elementos intencionais, que indica o grau de dependência das chamadas de serviços do documento AXML; e
- a altura do aninhamento de dependências de invocação, que determina o caminho crítico absoluto do documento.

Nestes cenários, é discutido o desempenho dos principais métodos de geração de planos de materialização, considerando: apenas o escopo de execução das chamadas de serviços (SE); a geração exaustiva das possibilidades de delegação de tarefas (CD); e o uso da heurística "Dividir para Conquistar" para quebrar o plano em tarefas independentes (D&C). Vale mencionar que no método D&C também é avaliada a delegação de chamadas de serviços, porém ela é restrita aos nodos de cada tarefa de materialização.

Observe que a análise do espaço de busca dispensa o conhecimento de informações sobre os resultados dos serviços Web referenciados em um documento AXML, pois o número de planos alternativos depende apenas da configuração do grafo de dependências e da estrutura do sistema P2P.

Os resultados discutidos a seguir têm ênfase na análise da seleção de sítios para a materialização AXML. Assim, por simplificidade de apresentação, não são consideradas as possíveis seqüências de invocações do documento AXML (*i.e.*, o agendamento de invocações dos planos de materialização).

6.3.1 VARIAÇÃO DO NÚMERO DE SÍTIOS

O número de planos físicos equivalentes de um documento AXML depende da quantidade de sítios que participam da materialização. Esses sítios podem ser provedores de serviços Web ou vizinhos do sítio mestre que estão disponíveis para colaboração P2P. Note que o número de sítios envolvidos em um documento AXML não representa o tamanho do sistema P2P. Por exemplo, um sistema pode ter cem sítios distintos, mas apenas cinco sítios (em média) são provedores de cada chamada de serviço de um documento AXML.

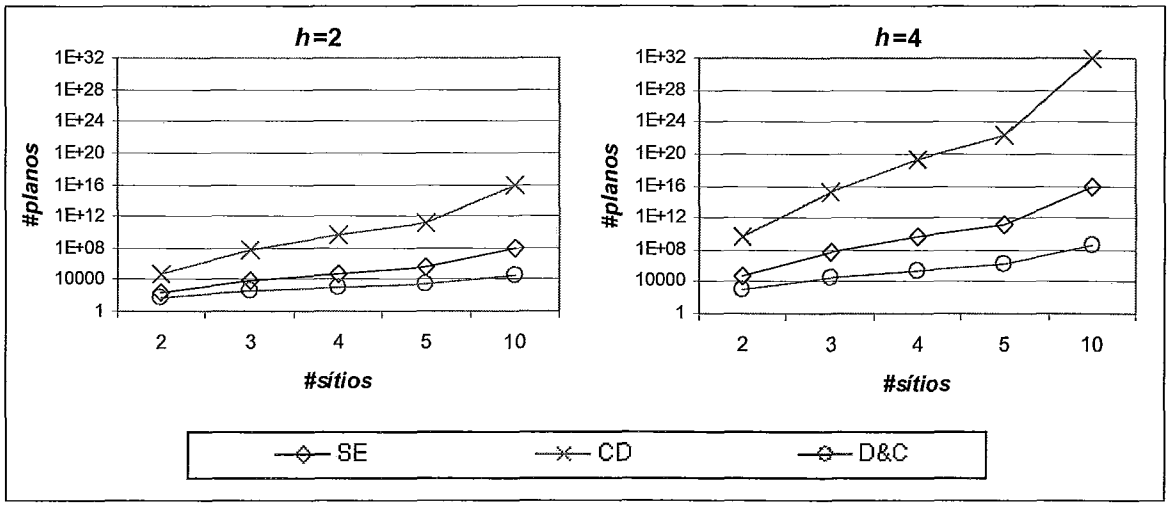


Figura 28. Crescimento do espaço de busca com a variação do número de sítios, considerando $|\Lambda|=4$ e *fanIn*=1.

A Figura 28 mostra resultados obtidos com a variação do número médio de sítios por chamada de serviço de um documento AXML, indicado por *#sítios*. Neste

caso, pressupõe-se que o número médio de provedores por chamada de serviço é igual ao número de possíveis controladores. O eixo de "*#planos*" está em escala logarítmica e representa o número de planos equivalentes que são enumerados na otimização, conforme o método de geração adotado. As chamadas de serviços têm *fanIn*=1 e o grafo de dependências usado nas simulações corresponde à configuração (*ii*). Além disso, considera-se que o número de tarefas independentes no documento AXML é $|\Lambda|=4$.

Os resultados da Figura 28 estão divididos para documentos cujas tarefas de materialização têm altura $h=2$ e $h=4$. Como esperado, em ambos os cenários acontece um crescimento substancial do número de planos com o aumento do número de sítios envolvidos na materialização. Esse crescimento se torna ainda mais expressivo em relação ao aumento da altura das tarefas de materialização. Mesmo quando é considerado apenas o escopo de execução (SE), o número de planos físicos equivalentes ainda pode se tornar muito grande. Por exemplo, para $h=4$ e *#sítios*=4, o método SE gera 4.294.967.296 planos. Supondo que cada plano é avaliado em 0.5*ms*, isso corresponde a mais de 24 dias (sic!) somente para selecionar o melhor plano físico.

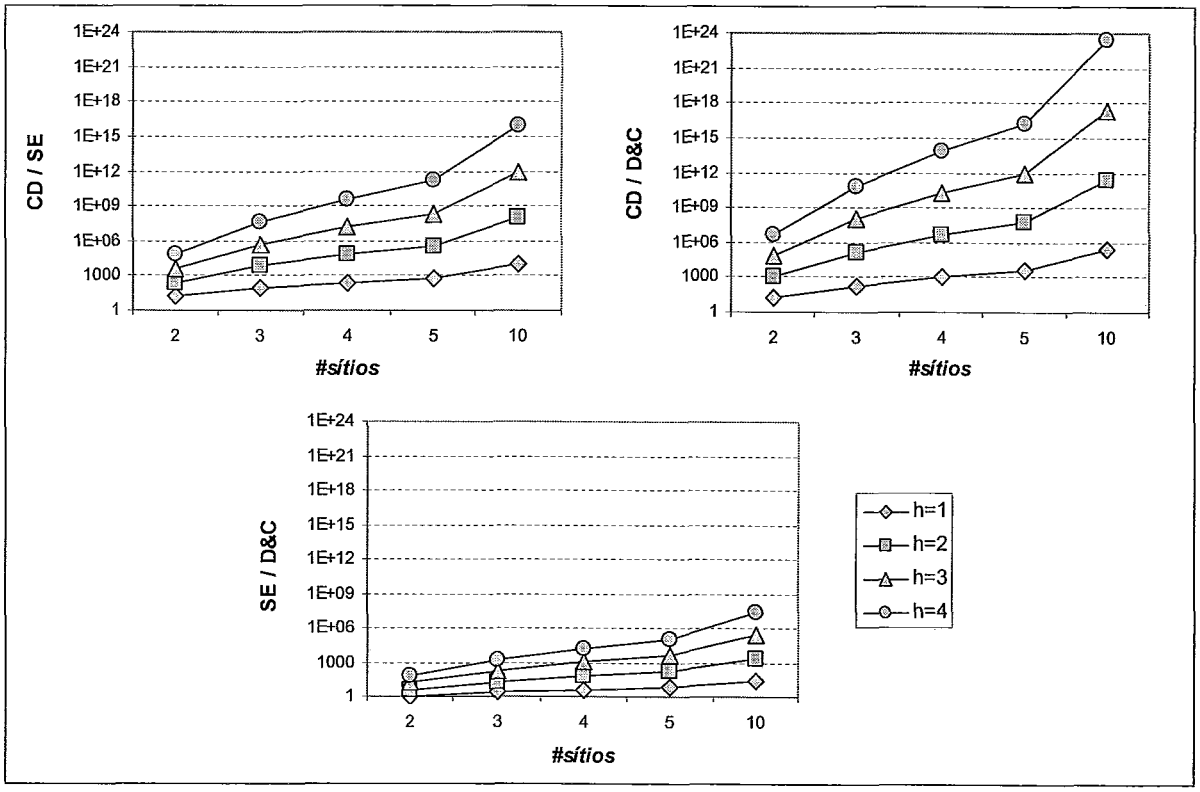


Figura 29. Comparação de estratégias na variação do número de sítios.

A Figura 29 reúne comparações relativas à redução do número de planos que são enumerados pelos diferentes métodos de geração, variando a altura do documento AXML. Note que quando h=1 o documento AXML corresponde ao padrão de "*bag of tasks*". O gráfico CD/SE mostra como o espaço de busca aumenta significativamente ao se considerar a delegação de chamadas de serviços. Além disso, entre os métodos analisados, a heurística D&C se mostra bastante interessante para reduzir o tamanho do espaço de busca. Porém, nem sempre é possível quebrar um plano de materialização em tarefas independentes, devido à presença de dependências compartilhadas. O otimizador XCraft consegue usar a heurística D&C para quebrar planos em tarefas mesmo quando há dependências compartilhadas, por meio de transformações do grafo.

Vale ressaltar que sistemas P2P geralmente envolvem uma quantidade bem maior de sítios do que os valores usados nestas simulações.

6.3.2 VARIAÇÃO DO *fanIn* DAS CHAMADAS DE SERVIÇOS WEB

Outro fator importante para o desempenho da materialização de um documento AXML é o número de parâmetros intencionais das chamadas de serviços (representado pelo parâmetro *fanIn*). Nesta simulação, foram usados documentos AXML com a configuração (*iii*), variando o *fanIn* das chamadas de serviços entre 2 e 5. Note que o *fanIn* pressupõe a existência de parâmetros intencionais. Portanto, não faz sentido avaliar a configuração "*bag of tasks*". Considera-se que *#sítios*=3 e |Λ|=4.

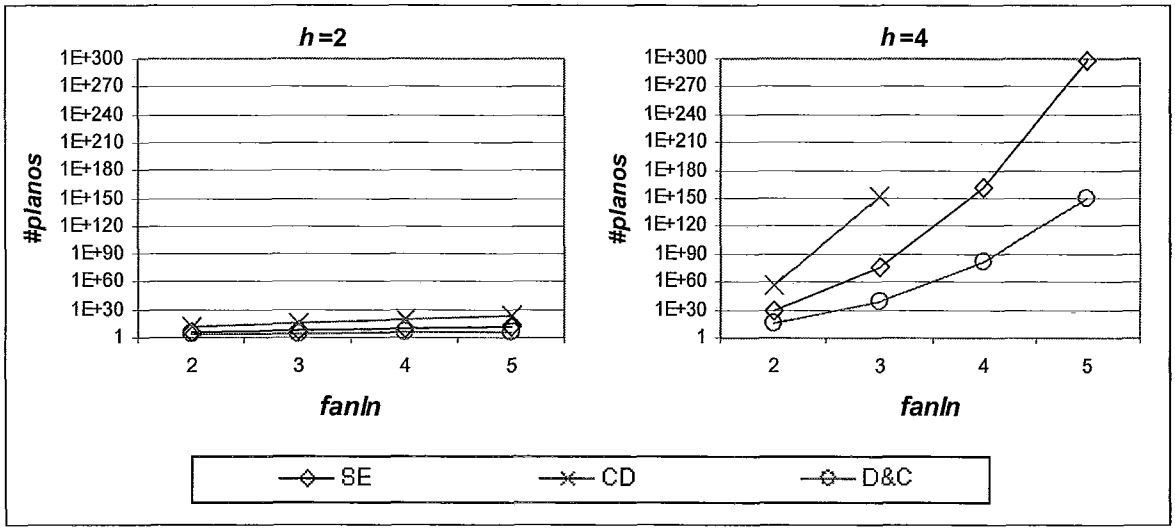


Figura 30. Crescimento do espaço de busca com a variação do número de parâmetros intencionais das chamadas de serviços, considerando *#sítios*=3 e |Λ|=4.

A Figura 30 mostra o número de planos equivalentes obtidos com a variação do *fanIn* para documentos com altura h=2 e h=4. Neste caso, percebe-se claramente que o aumento da altura tem um efeito substancialmente mais forte em comparação à variação do número de sítios. Isso ocorre porque variar o *fanIn* aumenta o número de nodos do grafo de dependências. Como todas as chamadas de serviços têm o mesmo *fanIn* nas simulações, o número de nodos do grafo de dependências é dado por:

$$|V| = \sum_{i=0}^{h-1} fanIn^h \quad . \tag{8}$$

Portanto, o *fanIn* tem impacto direto no tamanho do problema de otimização. Todavia, o aumento do número de chamadas de serviços não implica necessariamente em um aumento do tamanho do conteúdo do documento AXML, pois chamadas aninhadas representam resultados intermediários. Neste caso, o otimizador pode obter ganhos relevantes de desempenho com a delegação de tarefas. Estes ganhos são ignorados pela estratégia SE. Observe que no gráfico de h=4 não há resultados para *fanIn*=4 e *fanIn*=5. Esta ausência se deve ao estouro de precisão numérica, devido ao crescimento exagerado dos respectivos espaços de busca.

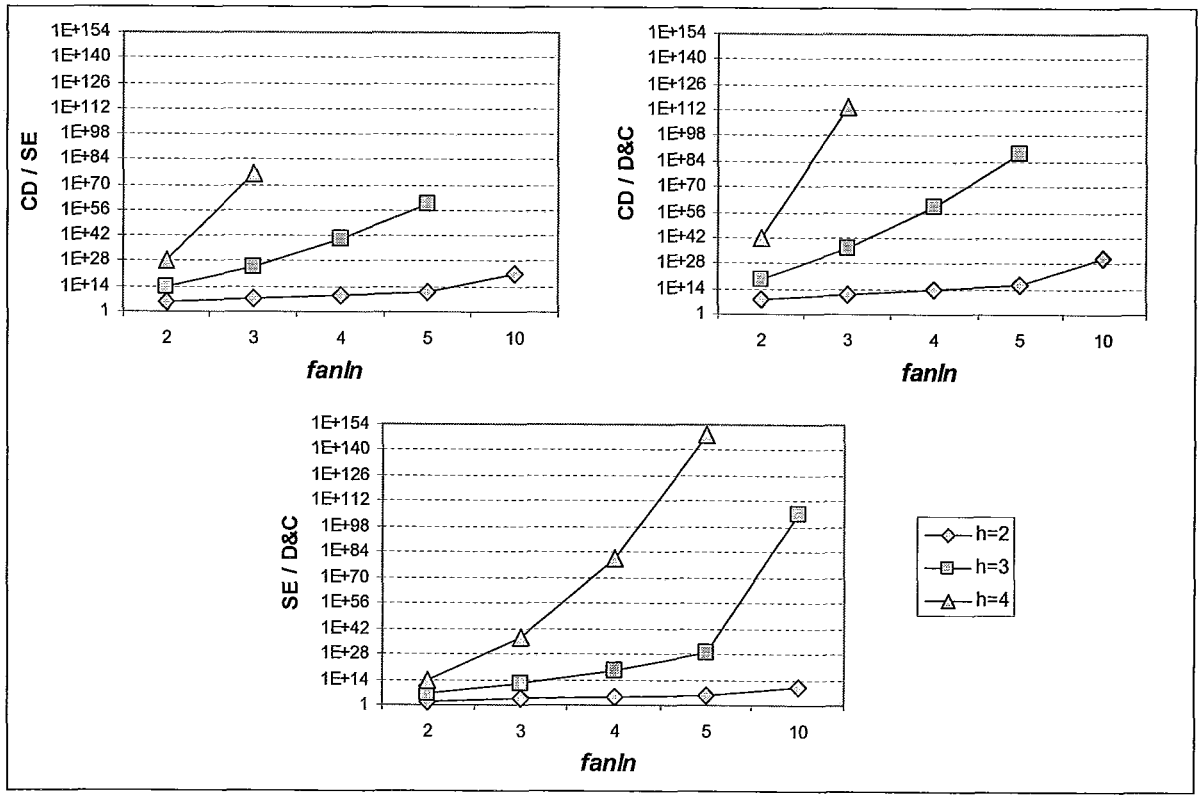


Figura 31. Comparação de estratégias na variação do *fanIn*.

Os resultados comparativos da variação do *fanIn* com os métodos de geração CD, SE e C&D são mostrados na Figura 31. Neste caso, o aumento brutal do espaço de busca reforça a vantagem das estratégias heurísticas. Os documentos usados nestas simulações não possuem dependências compartilhadas, favorecendo a aplicação da heurística D&C. Por outro lado, vale mencionar que o compartilhamento de resultados de invocações ajuda a reduzir o número de chamadas de serviços do documento AXML. O contrário ocorre com chamadas colaterais, que representam novas invocações de serviços Web.

Em casos extremos, para reduzir o *fanIn* médio de um plano de materialização, o otimizador XCraft pode usar os operadores **pipe** e θ para dividir os filhos dos operadores do plano. Note que esses operadores também podem ser executados pelo próprio sítio mestre.

6.3.3 Geração Dinâmica de Planos de Materialização

Na estratégia de otimização proposta, o XCraft quebra tarefas de materialização em subplanos de altura menor. Com isso, o otimizador consegue reduzir bastante a complexidade do problema e ainda considerar fluxos de dados entre chamadas de serviços do documento AXML, visando explorar a delegação de invocações.

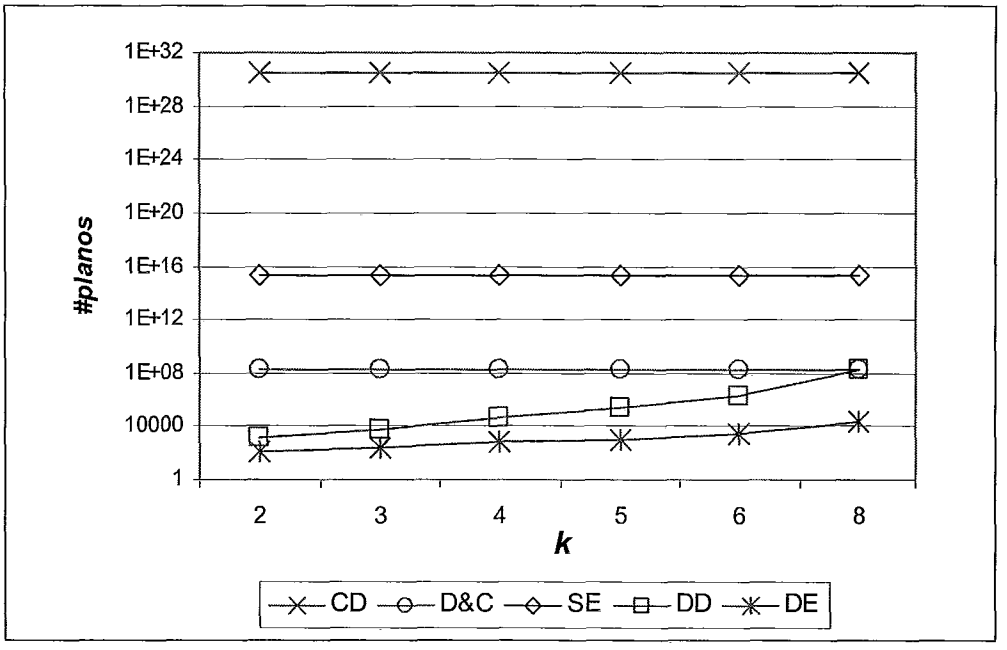


Figura 32. Número de planos gerados em diferentes estratégias de otimização para *h*=8, *fanIn*=1, *#sítios*=3 e |Λ|=4, variando o fator de quebra.

A Figura 32 mostra o desempenho de diferentes estratégias de otimização para um documento AXML da configuração (*ii*) com altura h=8, supondo que *#sítios*=3 e $|\Lambda|$=4. As estratégias DE e DD correspondem à estratégia dinâmica do XCraft com a geração de planos considerando, respectivamente, somente o escopo de execução e a delegação de chamadas de serviços.

Os resultados são obtidos pela variação do fator de quebra k entre 2 e 8. Por exemplo, para k=2, o otimizador quebra cada tarefa de materialização em quatro subplanos com mesma altura. Neste caso, para valores ímpares de k, o otimizador gera subplanos com alturas diferentes. Podemos verificar que o XCraft permite estabelecer um limite bem interessante para o tamanho do espaço de busca, comparado às estratégias CD e SE.

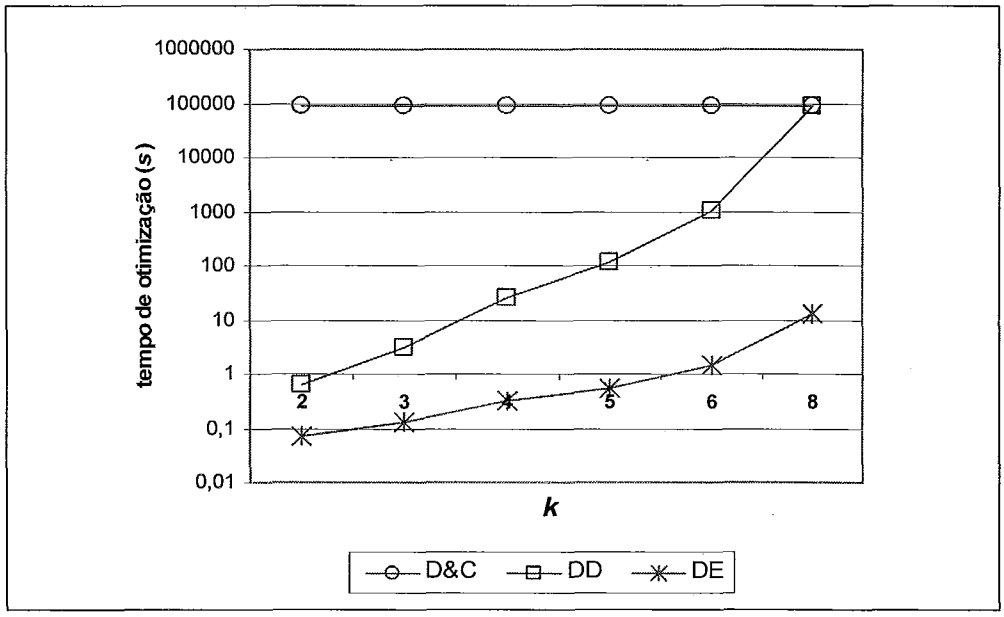


Figura 33. Tempo de otimização de diferentes estratégias de otimização.

A geração de planos físicos equivalentes representa a principal etapa de uma estratégia de otimização. Assim, a partir do número de planos físicos gerados, é possível estimar o tempo gasto na otimização da materialização de um documento AXML. Em experimentos reais, observamos que o tempo de otimização médio por plano é aproximadamente 0.5*ms* no sítio 1. Considerando esse valor, a Figura 33 mostra os tempos de otimização estimados (em segundos) para as estratégias D&C, DD e DE. Observe que a partir de k=5, o tempo gasto em otimização se torna significativo. Porém,

para $k < 5$, o XCraft ainda pode explorar a delegação para evitar a transferência de resultados intermediários em até três níveis dos subplanos de materialização.

Vale destacar que os planos gerados pelas estratégias DE e DD são menores que os gerados pelas estratégias CD, SE e D&C. Portanto, o XCraft tende a gastar ainda menos tempo de otimização em relação às demais estratégias. Além disso, o XCraft produz resultados parciais, beneficiando também o tempo de resposta para os primeiros resultados.

6.4 Efeitos da Delegação de Chamadas de Serviços

Por ser baseada em heurísticas, a estratégia de otimização do XCraft não produz soluções ótimas. Contudo, ela adota uma abordagem descentralizada que permite reduzir os custos de transferências de dados, os quais geralmente têm um peso significativo em redes P2P heterogêneas. Com o protótipo do XCraft, foram realizados testes experimentais para avaliar o desempenho da abordagem baseada na delegação de tarefas em relação à abordagem centralizada (*i.e.*, quando o sítio mestre invoca todas as chamadas de serviços do documento AXML).

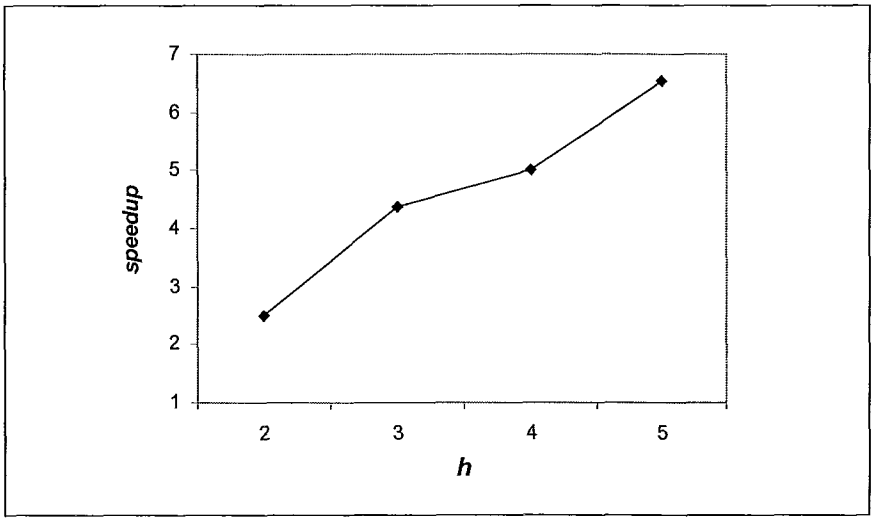


Figura 34. Ganho de desempenho obtido pela delegação de tarefas no XCraft, em relação à abordagem centralizada.

A Figura 34 mostra o ganho de desempenho da estratégia baseada em delegação de tarefas em relação à abordagem centralizada, variando a altura das tarefas de materialização entre 2 e 5. Considera-se que $|\Lambda|=32$ e *fanIn*=2. As chamadas de serviços

retornam um resultado com 500K*bytes*. Na estratégia com delegação, cada tarefa de materialização é executada remotamente e apenas os resultados persistentes são transferidos para o sítio mestre.

6.5 GERAÇÃO DE PLANOS POR MEIO DE BUSCA LOCAL

Em PEREIRA, RUBERG e MATTOSO (2005), o otimizador XCraft foi estendido com uma estratégia chamada de SLS-MC (de "*Stochastic Local Search with Multiple stop Conditions*"). Esta estratégia explora um método de busca local estocástica e diferentes heurísticas para agendamento da invocação de chamadas de serviços. Métodos estocásticos costumam ser bastante interessantes para lidar com cenários muito complexos, nos quais a geração exaustiva não é possível. Isto acontece com freqüência na materialização de documentos AXML.

Resultados da avaliação experimental da SLS-MC são discutidos em PEREIRA, RUBERG e MATTOSO (2006). O testes consideram três documentos AXML com 33, 65 e 129 chamadas de serviços. Nesses documentos, as chamadas de serviços estão aninhadas com altura h=3 e cada chamada de serviço tem *fanIn*=3. Além disso, são considerados 15 serviços Web distribuídos em 4 sítios de uma rede P2P heterogênea. Na geração de planos físicos, são avaliadas heurísticas típicas para agendamento de tarefas em *workflows*: MET (de "*Minimum Execution Time*"), que aloca cada operador ao provedor que oferece o melhor tempo de execução; min-min, que identifica o menor tempo de conclusão de cada operador do plano e aloca tarefas menores primeiro; max-min, que aloca tarefas maiores primeiro; e duplex, que aplica as heurísticas min-min e max-min para escolher o melhor resultado entre as duas.

A Figura 35 apresenta resultados experimentais obtidos com a SLS-MC, relativos à análise das heurísticas de agendamento. A condição de parada da estratégia é fixada pelo número máximo de planos avaliados, calculado em 5% do espaço de planos equivalentes. Os resultados da Figura 35(a) correspondem ao custo de planos gerados por um método guloso, para cada heurística de agendamento de tarefas. Esses planos são refinados pela SLS-MC até que a condição de parada seja atingida. O custo final de cada plano é mostrado na Figura 35 (b).

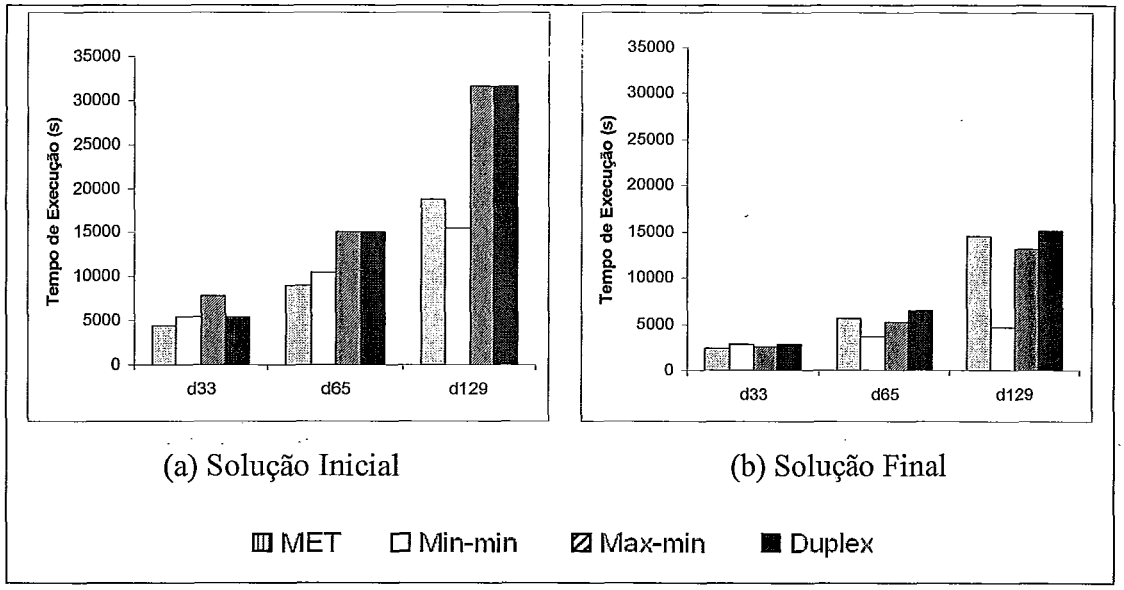


Figura 35. Resultados da estratégia SLS-MC com diferentes heurísticas de agendamento de invocações.

Observe que, nos resultados da Figura 35, a SLS-MC consegue reduzir em média 50% os tempos de execução obtidos por métodos gulosos, independente da heurística de agendamento utilizada. O desempenho da SLS-MC também foi avaliado em diferentes condições de parada (PEREIRA, RUBERG e MATTOSO, 2006). Em geral, os resultados experimentais da SLS-MC indicaram ganhos importantes na otimização da materialização AXML.

6.6 FRAGMENTAÇÃO DE DOCUMENTOS XML

As técnicas de otimização propostas no XCraft tratam da materialização de documentos AXML, com enfoque no desempenho da invocação das chamadas de serviços e nos problemas de dinamismo e descentralização dos sistemas P2P. Por outro lado, o volume dos dados armazenados também tem impacto no desempenho do acesso ao conteúdo de um documento AXML. Em contexto mais abrangente, as tradicionais técnicas de fragmentação de dados podem ser muito úteis para reduzir a quantidade de dados manipulados pelas consultas que acessam o repositório de documentos XML. Entretanto, a aplicação dessas técnicas em documentos AXML não é trivial, devido a algumas características específicas do modelo de dados XML, tais como o aninhamento de elementos e e sofisticação da projeção de nodos de uma árvore.

Em uma proposta complementar ao XCraft, esta contribuiu para a definição de um modelo de fragmentação de dados XML, chamado **PartiX** (ANDRADE, RUBERG *et al.*, 2005, 2006). Esse modelo define as principais técnicas para a fragmentação de dados XML, abrangendo tanto fragmentação horizontal como vertical. Ele é baseado na álgebra de operadores TLC (PAPARIZOS *et al.*, 2004). Além disso, o PartiX considera repositórios representados por um único documento XML ou por múltiplos documentos. Resultados experimentais com o PartiX são discutidos em ANDRADE, RUBERG *et al.* (2006).

Capítulo 7

CONCLUSÕES E TRABALHOS FUTUROS

Este capítulo reporta as conclusões gerais e contribuições desta tese, bem como as principais perspectivas de trabalhos futuros.

7.1 CONSIDERAÇÕES FINAIS

Nos últimos anos, uma forte tendência tem impulsionado as aplicações computacionais a se tornarem cada vez mais distribuídas e descentralizadas. Seja pela crescente necessidade de colaboração entre pessoas e instituições, ou pela escalada da complexidade dos problemas tratados, grande parte dos sistemas de hoje já não pode mais se limitar a um único supercomputador ou mesmo a uma rede privada. O desenvolvimento dessas aplicações tem sido favorecido por um arcabouço de padrões que constituem a denominada Web 2.0 (O'REILLY, 2005). Indo além, recentemente esse arcabouço tem sido explorado na construção de camadas semânticas para integrar recursos na Web 3.0 (SHANNON, 2006). Tais denominações se referem às novas gerações de aplicações e serviços baseados em tecnologias da Web (*i.e.*, HTTP, CSS, serviços Web, etc.). Entre os precursores deste contexto, os documentos XML ativos – ou AXML – surgiram como uma mídia capaz de incorporar conteúdo advindo de diversas fontes de dados, em uma abordagem inovadora e flexível que combina técnicas de integração tanto física como virtual.

Em documentos AXML, a tecnologia de serviços Web é explorada como uma "cola" para unir dados e programas distribuídos (ABITEBOUL *et al.*, 2002, 2003b). Certos elementos de um documento AXML são ditos intencionais pois representam chamadas de serviços Web cujos resultados devem ser materializados para compor o conteúdo do documento. A idéia de embutir chamadas de serviços Web em documentos AXML cria muitas possibilidades interessantes. Por exemplo, essa combinação permite que invocações de programas sejam manipuladas declarativamente. Além disso, elementos intencionais também permitem distribuir tarefas dinamicamente por intermédio de serviços Web que retornam respostas ativas (*i.e.*, que contêm outras chamadas de serviços). Contudo, como a materialização de documentos AXML ocorre tipicamente em sistemas P2P, o desempenho desse processo depende de um número considerável de fatores, podendo variar bastante entre alternativas de avaliação.

Os relacionamentos entre elementos intencionais de um documento AXML causam restrições nas invocações de serviços Web, as quais correspondem a padrões de fluxos de controle e de dados em *workflows* (AALST *et al.*, 2003). Vale mencionar que em ambientes distribuídos, tais como sistemas P2P e grades computacionais (*grids*), a

necessidade de técnicas de otimização para a execução de *workflows* é notoriamente reconhecida na literatura (MILOJICIC *et al.*, 2002, YU e BUYYA, 2005). Para suprir plenamente essa carência, alguns desafios precisam ser superados, sobretudo em relação à descentralização da execução e do controle de processos inter-relacionados. Também devem ser tratados os problemas decorrentes da heterogeneidade e do dinamismo dos recursos nesses sistemas.

Embora existam várias propostas de otimizadores para a execução de *workflows* distribuídos, principalmente em *grids*, os trabalhos existentes são baseados em abordagens centralizadas. Muitos desses otimizadores são estáticos (ALHUSAINI *et al.*, 1999, BAUER e DADAM, 1997, BLYTHE *et al.*, 2005, BRAUN *et al.*, 2001, CAO *et al.*, 2003, GOUNARIS *et al.*, 2004b, VARGAS *et al.*, 2005, WIECZOREK *et al.*, 2005, ZENG *et al.*, 2004, ZHANG *et al.*, 2006), enquanto as abordagens dinâmicas utilizam algoritmos gulosos (OINN *et al.*, 2004, THAIN *et al.*, 2005, TRIANA, 2003, TAYLOR *et al.*, 2003) ou oportunistas (MEYER *et al.*, 2006b, SINGH *et al.*, 2006). Assim, esses otimizadores costumam ignorar benefícios importantes da delegação de tarefas, pois não analisam fluxos de dados entre processos.

Por outro lado, otimizadores que exploram algum grau de descentralização (ARORA *et al.*, 2002, BAUER e DADAM, 1997, 2000, BENATALLAH *et al.*, 2005, CAO *et al.*, 2003, MUTH *et al.*, 1998, VARGAS *et al.*, 2004, 2005, WODTKE *et al.*, 1996) tratam apenas a delegação do controle de tarefas e pressupõem que a seleção dos sítios é arbitrária, sendo baseada em alguma heurística ou determinada pelo usuário. Esses otimizadores limitam os sítios controladores a um conjunto fixo pré-determinado. Em geral, o processo de otimização é centralizado e não leva em conta custos substanciais da invocação de serviços Web (ALHUSAINI *et al.*, 1999, AZEVEDO, 2003, BAUER e DADAM, 2000, BENATALLAH *et al.*, 2005, BLYTHE *et al.*, 2005, YU *et al.*, 2005, ZENG *et al.*, 2004). Além disso, esses otimizadores não são capazes de lidar com alterações na especificação de um *workflow* em tempo de execução, dependendo essencialmente de re-otimizações que podem invalidar totalmente planos previamente estabelecidos.

Para fundamentar a estratégia de otimização proposta nesta tese, consideramos inicialmente o modelo de documentos AXML apresentado em ABITEBOUL *et al.* (2002, 2003b) e MILO *et al.* (2003). Porém, observamos que a tradicional estrutura de

árvore usada nestes trabalhos não é adequada para expressar as restrições de invocação e os fluxos de dados entre as chamadas de serviço de um documento AXML durante o processo de materialização. Assim, propomos um formalismo inspirado em modelos para *workflows* (AALST *et al.*, 2003, EROL *et al.*, 1994, LIU *et al.*, 2004b), acrescentando características relevantes para a materialização AXML. O formalismo proposto representa documentos AXML por meio de grafos de dependências e inclui regras para simplificação de redundâncias baseadas no conceito de equivalência entre grafos, em uma abordagem semelhante à adotada em SUCIU (2002). Como no sistema Vortex (HULL *et al.*, 2000), esse formalismo realça no documento AXML os elementos intencionais cujos resultados são persistentes (*i.e.*, que compõem o conteúdo do documento AXML). Esse destaque é importante para reduzir as transferências de dados na materialização AXML pela colaboração P2P.

Nesta tese, um algoritmo dinâmico gera planos de materialização AXML a partir de grafos de dependências. Como no sistema D2WMS (DIAS *et al.*, 2003) para a execução de *workflows*, o algoritmo proposto realiza execuções parciais. Ou seja, documentos AXML são materializados incrementalmente. Porém, no D2WMS cabe ao usuário determinar a divisão das partes de cada *workflow*, enquanto planos de materialização AXML são divididos automaticamente no XCraft.

As dependências compartilhadas e as chamadas colaterais de um documento AXML causam acoplamento de dados e programas entre os componentes no grafo de dependências, dificultando a colaboração entre sítios. Assim, para simplificar a estrutura do grafo de dependências na geração de planos de materialização, esta tese propõe um algoritmo de destacamento de nodos (por replicação e por clonagem) que transforma o grafo em uma floresta mínima de árvores geradoras. Esta floresta consiste em uma estrutura mais adequada para execuções distribuídas. O algoritmo explora um *cache* de chamadas de serviços para lidar com o compartilhamento de dependências de invocação. Ele foi inspirado em técnicas clássicas para decomposição de grafos de consultas (ELMASRI e NAVATHE, 1994). Na geração de planos também foram consideradas as principais heurísticas de agendamento de tarefas (BRAUN *et al.*, 2001, DONG e AKL, 2006).

Testes experimentais foram realizados com um protótipo do otimizador XCraft, construído sobre o sistema *ActiveXML* (ACTIVEXML, 2002), uma plataforma P2P de

software livre para a gerência de documentos AXML. Para alimentar os serviços Web usados nos testes, foram gerados documentos XML sintéticos do *benchmark* XMark (XMARK, 2003) com o auxílio da ferramenta ToxGene (BARBOSA *et al.*, 2002).

7.2 CONTRIBUIÇÕES

A principal contribuição desta tese consiste em uma estratégia descentralizada e dinâmica para otimizar a materialização de documentos AXML em sistemas P2P. Vários problemas são tratados na estratégia proposta, que é realizada pelo otimizador XCraft. Inicialmente, introduzimos um modelo canônico para representar os relacionamentos entre chamadas de serviços de um documento AXML. Este formalismo abrange regras de validade quanto a ciclos maliciosos (*i.e.*, que causam bloqueios perpétuos e ciclos infinitos de execução) e mecanismos para a atualização de grafos de dependências com respostas ativas. A partir deste formalismo, são descritas as principais alternativas de execução e colaboração P2P na materialização de um documento AXML, contemplando o uso de referências abstratas de serviços Web. O XCraft considera a delegação tanto do controle como da própria otimização de planos de materialização.

Nesta tese, caracterizamos o problema de otimização de desempenho da materialização AXML e identificamos os principais limites de complexidade da busca por soluções eficientes. São analisados diferentes métodos de geração de planos, com destaque para: a busca exaustiva, com a enumeração de todas as possíveis combinações de recursos e de agendamento de chamadas de serviços de um documento AXML; o uso da tradicional heurística de dividir para conquistar, que quebra planos de materialização em partes independentes; e a busca gulosa, que faz a seleção dos sítios pelo melhor tempo de execução de cada chamada de serviço. Além disso, a estratégia proposta permite o uso de heurísticas para limitar os sítios candidatos à colaboração P2P, a partir do conjunto de possíveis executores de serviços de um plano de materialização. Vale lembrar que o problema de determinar um mapeamento eficiente de processos em máquinas heterogêneas é reconhecidamente NP-completo, tornando imperativo o uso de heurísticas. No XCraft, adota-se uma meta-heurística que permite selecionar o método de geração adequado para cada plano de materialização, podendo ainda combinar diferentes métodos em um plano.

A estratégia de otimização do XCraft é baseada em algoritmo dinâmico que quebra automaticamente um grafo de dependências em partes menores, reduzindo a complexidade da busca por planos eficientes e favorecendo a descentralização da materialização AXML. São intercaladas etapas de otimização e execução. O algoritmo posterga a localização de serviços Web, facilitando a adaptação de planos de materialização ao dinamismo dos sítios em um sistema P2P. Planos são divididos em tarefas de materialização, que por sua vez são quebradas em subplanos de altura pré-determinada. São apresentados mecanismos para atribuir prioridades às tarefas de um plano, baseados em um conjunto de critérios que abrange o acoplamento de dados decorrente de dependências compartilhadas. Esses mecanismos visam aumentar o potencial de execução paralela entre tarefas durante a avaliação de um plano de materialização.

Uma peça chave da geração de planos de materialização no XCraft consiste em uma álgebra de operadores que introduz várias inovações importantes. Esta álgebra possui operadores que tratam explicitamente da delegação de chamadas de serviços e da descentralização da otimização. Assim, o otimizador pode solicitar a sítios vizinhos que colaborem para a materialização de um documento AXML, bem como para a própria geração de um determinado plano. A álgebra do XCraft se baseia em um conjunto de serviços Web básicos para colaboração P2P. Além disso, ela inclui operadores lógicos que permitem uma avaliação incremental dos planos. Esta abordagem é extensível e também pode se beneficiar de técnicas tradicionais de otimização algébrica.

Outra contribuição desta tese consiste em um modelo de custo para estimar o desempenho de planos de materialização equivalentes. Uma novidade relevante deste modelo analítico é que ele oferece três níveis de estimativa. O primeiro nível se aplica a planos abstratos e permite estimar a complexidade da otimização, ajudando na escolha do método de geração de planos físicos equivalentes. Já no segundo nível, o otimizador pode estimar o custo de planos parciais por meio de uma análise heurística. O terceiro nível trata do cálculo detalhado e abrangente dos custos de planos de materialização, considerando a ordem de execução dos operadores, sistemas P2P heterogêneos e a carga de tarefas atribuídas aos sítios. A análise de cada plano é baseada em um conjunto de funções para estimar os custos básicos de diferentes estratégias para a materialização AXML. Em especial, essas funções levam em conta operações típicas da invocação de

serviços Web, tais como a serialização de dados XML e o empacotamento de mensagens SOAP (RUBERG *et al.*, 2004a). Esses custos são ignorados pelos otimizadores encontrados na literatura.

A análise de desempenho desta tese utiliza um conjunto de estatísticas e parâmetros de custo sobre os documentos AXML, os serviços Web e o sistema P2P. Para coletar essas informações auxiliares, consideramos que sítios publicam um conjunto de serviços Web básicos e documentos AXML de calibração. Esta proposta resultou da pesquisa realizada inicialmente com o DIG (de "*Distributed Information Gatherer*"), um provedor de custos e estatísticas em ambientes distribuídos, heterogêneos e com fontes de dados autônomas (RUBERG *et al.*, 2002, 2003). O DIG é baseado em uma arquitetura de mediadores que mantém um catálogo de informações e oferece serviços para a aquisição e a publicação das características gerais, custos e estatísticas de diferentes fontes de dados. O DIG foi apresentado na dissertação de mestrado de Nicolaas Ruberg (RUBERG, 2002).

As técnicas propostas nesta tese foram desenvolvidas no XCraft (RUBERG e MATTOSO, 2007a, 2007b), um otimizador baseado em uma arquitetura descentralizada e orientada a serviços Web. Para avaliar o impacto da estratégia de otimização no desempenho da materialização de documentos AXML, implementamos um protótipo do XCraft na plataforma *ActiveXML* (ACTIVEXML, 2002). Os resultados experimentais obtidos com esse protótipo mostraram que a estratégia dinâmica do XCraft consegue reduzir significativamente o número de planos equivalentes avaliados. Além disso, apesar de usar uma meta-heurística sub-ótima, o XCraft consegue explorar ganhos de desempenho substanciais com a delegação de chamadas de serviços Web.

Na avaliação experimental do XCraft, também analisamos o impacto do tamanho dos subplanos gerados em cada passo de otimização dinâmica. Dependendo do *fanOut* dos operadores de um subplano e da configuração do sistema P2P, observamos que pode ser percorrido um espaço de alternativas enorme mesmo quando são gerados subplanos de altura pequena (*e.g.*, 3 ou 4). Isso se deve sobretudo à natureza combinatória da geração de planos e praticamente inviabiliza a aplicação de estratégias exaustivas. Por outro lado, métodos puramente gulosos também são inadequados, pois são baseados em decisões locais que ignoram a delegação de chamadas de serviços. Vale destacar também que a delegação de tarefas implica que subplanos ótimos não

garante um plano ótimo. Ou seja, o custo de cada operador de um plano de materialização depende do arranjo global dos demais operadores. Para lidar com esses problemas, investigamos diferentes alternativas para a geração de planos de materialização. Em PEREIRA, RUBERG e MATTOSO (2006), estendemos o XCraft com um método de busca local estocástica para a geração eficiente de planos físicos de materialização, em uma estratégia chamada de SLS-MC (de "*Stochastic Local Search with Multiple stop Conditions*"). Com a SLS-MC, é possível estabelecer limites para o tempo gasto durante a otimização. Ela permite, durante a geração de planos equivalentes, que o otimizador aceite planos de desempenho inferior para fugir de mínimos locais. A aceitação de planos é controlada por um fator de probabilidade arbitrário. Para avaliar a SLS-MC, o otimizador XCraft foi encapsulado em um ambiente de simulação chamado SiMAX, desenvolvido no contexto da dissertação de mestrado de Daniela Pereira (PEREIRA, RUBERG e MATTOSO, 2005, PEREIRA, 2007). Com o SiMAX, verificamos que a SLS-MC mostrou excelentes resultados com diferentes heurísticas de agendamento de tarefas.

Nesta tese também foi desenvolvido o sistema *Acware* (ABITEBOUL, RUBERG e NGUYEN, 2006b), que permitiu evidenciar muitas questões de desempenho envolvidas na materialização AXML. O *Acware* utiliza documentos AXML para construir repositórios de conteúdo ativo. Por intermédio de uma linguagem gráfica, são exploradas consultas sobre documentos AXML para gerar as chamadas de serviços que alimentam e enriquecem um repositório de conteúdo. Desenvolvemos um protótipo do *Acware* em um projeto para controle de risco alimentar do Ministério de Agricultura da França. Os resultados obtidos tornaram patente a necessidade do uso de referências abstratas de serviços Web em documentos AXML, bem como de métodos automáticos para a geração de planos de materialização. Em AZIS, RUBERG *et al.* (2004), estendemos a arquitetura do *Acware* com um módulo de consultas chamado *Acware Viewer*. Este módulo adota uma estratégia baseada em ontologias para a classificação e a apresentação de dados de um repositório de conteúdo. Um protótipo do *Acware Viewer* foi apresentado como trabalho de graduação de Nida Azis na *École Polytechnique*, na França.

Complementar ao XCraft, entre as contribuições desta tese está o modelo de fragmentação de dados XML do PartiX (ANDRADE, RUBERG *et al.*, 2005, 2006).

Esse modelo permite aplicar técnicas tradicionais de fragmentação em bases de dados XML, baseado na álgebra de operadores TLC (PAPARIZOS *et al.*, 2004). O PartiX contempla as principais características dos dados XML e trata adequadamente repositórios com múltiplos documentos. Isso pode melhorar bastante a execução de consultas XQuery em bases volumosas, conforme mostram os resultados obtidos em ANDRADE, RUBERG *et al.* (2006). Testes experimentais com o PartiX foram realizados na dissertação de mestrado de Alexandre Andrade (ANDRADE, 2006). Vale mencionar que o PartiX contribuiu para o desenvolvimento de outros trabalhos neste tema, tanto no grupo de Banco de Dados da COPPE/UFRJ (FIGUEIREDO, 2007, FIGUEIREDO *et al.*, 2007) como em outras instituições (SOUSA e MACHADO, 2006, SOUSA *et al.*, 2007a, 2007b).

Ao longo desta tese ocorreram algumas colaborações de pesquisa cujas contribuições também merecem destaque. Em particular, foi desenvolvido o tradutor XVerter (VIEIRA, RUBERG e MATTOSO, 2002, 2003) para converter consultas escritas na linguagem XQuery em comandos da OQL, uma linguagem típica de SGBDs baseados em objetos. O XVertex explora uma camada genérica de persistência baseada no DOM (*Document Objet Model*), favorecendo a convivência de dados XML com dados de sistemas legados. Ele define um conjunto de regras de transformação que cobrem as principais características da linguagem XQuery. O XVerter foi apresentado na dissertação de mestrado de Humberto Vieira (VIEIRA, 2002). Também vale ressaltar a participação na especificação do modelo de custos para seleção de serviços Web em uma arquitetura com controle transacional, na dissertação de mestrado de Valdino Azevedo (AZEVEDO, 2003). Recentemente, esta tese contribuiu na definição da arquitetura do sistema ARAXA (FERRAZ *et al.*, 2006, 2007), que trata do armazenamento de documentos AXML em SGBDs baseados em objetos.

7.3 TRABALHOS FUTUROS

Em um contexto mais amplo, a materialização de documentos AXML se assemelha à execução de *workflows* distribuídos, que é alvo de uma vasta gama de temas de pesquisa. Portanto, a estratégia de otimização do XCraft pode ser enriquecida com várias técnicas complementares. Por exemplo, técnicas de *pipeline* permitem reduzir os requisitos de espaço para armazenamento de dados na execução de

workflows. Para tratar esse aspecto, o XCraft deve considerá-lo no agendamento das chamadas de serviços Web, como apresentado em LEMOS e CASANOVA (2006). Vale ressaltar que o XCraft explora a proximidade entre sítios para reduzir o custo das transferências de dados. Isso favorece sobremaneira o uso de técnicas de *pipeline*.

Outra alternativa de continuação desta tese é antecipar, na geração de planos de materialização, o tratamento de falhas nas chamadas de serviços de um documento AXML. Para isso, o XCraft pode gerar planos com opções de contingência, como proposto em COSTA *et al.* (2004) para a composição de serviços Web. Ainda no contexto de serviços Web, seria interessante estender o XCraft para considerar contratos de colaboração P2P, como definido em PINHEIRO *et al.* (2007), ao selecionar coordenadores para a delegação de tarefas.

Embora tenha sido construído para tratar especificamente de documentos AXML, as técnicas de otimização do XCraft lidam com padrões básicos de *workflows*. Portanto, o XCraft pode ser estendido para incorporar uma máquina de execução para linguagens de *workflows* de serviços Web, como a WS-BPEL (OASIS, 2007). Outra extensão útil consiste em acrescentar um monitor gráfico ao XCraft, para que o usuário possa acompanhar a evolução dos planos processados pelo otimizador. Também poderiam ser introduzidos mecanismos interativos no XCraft, permitindo que o usuário interfira nos planos gerados.

Com a ferramenta SiMAX, o XCraft pode ser utilizado para auxiliar o projeto de alocação de serviços Web em uma rede P2P. Analogamente, uma aplicação bastante interessante do PartiX seria na construção de uma ferramenta para apoio ao projeto de distribuição de dados XML. A idéia consiste em encapsular o PartiX em um ambiente de simulação e usar o modelo de custo proposto em RUBERG (2001) e RUBERG *et al.* (2002a) para avaliar diferentes alternativas de fragmentação e alocação de dados. Tal ferramenta também seria útil para dimensionar recursos para a gerência de bases de documentos XML, tais como espaço em disco e tamanho do *cache* dados.

Por fim, as técnicas descentralizadas e orientadas a serviços do XCraft representam um passo importante para tratar da otimização de problemas complexos em ambientes distribuídos. Este é um desafio crucial para a consolidação das novas gerações de aplicações da Web, bem como em sistemas P2P e *grids*. Inspirados no

modelo de funcionamento do cérebro humano, estamos investigando uma generalização da arquitetura do XCraft para atacar esse tema. Essa arquitetura é chamada de rede glial[1]. No cérebro humano, células gliais exercem um papel fundamental no apoio ao sistema nervoso central, pois regulam o funcionamento interno dos neurônios e facilitam a comunicação entre eles. Na nossa proposta, as células de uma rede glial são agentes espalhados em um sistema P2P, os quais colaboram entre si para otimizar a realização de tarefas. Essa colaboração é dinâmica e descentralizada, sendo desempenhada por diferentes tipos de células gliais, que podem ser especializadas de acordo com as tarefas desempenhadas (*e.g.*, localização de serviços, aquisição de custos, controle da execução de tarefas, geração de planos, etc.). Além disso, uma rede glial é caracterizada por alguns processos básicos que guiam o funcionamento de suas células. Em particular, o processo de homeostase diz respeito à manutenção do equilíbrio dinâmico de cada sítio em relação ao desempenho. Cabe às células gliais monitorar um conjunto de indicadores do funcionamento de um sítio, disparando mecanismos auto-reguladores para corrigir eventuais desequilíbrios. Observe que a homeostase é um processo essencial em sistemas com participantes autônomos. Outro processo importante consiste na gliose, que provoca uma proliferação de células gliais em determinadas regiões da rede, visando compensar alguma deficiência no local. No modelo biológico, isso geralmente ocorre devido a alguma lesão no cérebro, ou em áreas sobrecarregadas. Por analogia, a gliose pode ajudar a balancear cargas de processamento e a corrigir falhas de sítios em uma rede glial.

Embora o XCraft implemente muitos dos mecanismos de uma rede glial, ainda são necessários protocolos adequados para controlar os processos descritos. Acreditamos que as redes gliais poderão ter um impacto muito positivo no processamento de tarefas em sistemas distribuídos.

[1] O termo *glia* vem do Grego e significa "cola".

REFERÊNCIAS BIBLIOGRÁFICAS

AALST, W.M.P., 1998, "The Application of Petri Nets to Workflow Management", *The Journal of Circuits, Systems and Computers*, v. 8, n. 1 (Fevereiro), pp. 21–66.

AALST, W.M.P., HOFSTEDE, A.H.M., KIEPUSZEWSKI, B. e BARROS, A., 2003, "Workflow Patterns", *Distributed and Parallel Databases*, v. 14, n. 1 (Julho), pp. 5–51.

ABITEBOUL, S., AMANN, B., CLUET, S., EYAL, A., MIGNET, L. e MILO, T., 1999, "Active Views for Eletronic Commerce". In: *Proceedings of the 25th International Conference on Very Large Data Bases (VLDB)*, pp. 138–149, Edimburgo, Escócia, Setembro.

ABITEBOUL, S., BENJELLOUN, O., MANOLESCU, I., MILO, T. e WEBER, R., 2002, "Active XML: Peer-to-Peer Data and Web Services Integration". In: *Proceedings of the 28th International Conference on Very Large Data Bases (VLDB)*, pp. 1087–1090, Hong Kong, China, Agosto.

ABITEBOUL, S., 2003, "Managing an XML Warehouse in a P2P Context". In: *Proceedings of the 15th International Conference on Advanced Information Systems Engineering (CAiSE)*, LNCS 2681, pp. 4–13, Klagenfurt, Áustria, Junho.

ABITEBOUL, S., AMANN, B., BAUMGARTEN, J., BENJELLOUN, O., DANG-NGOC, F. e MILO, T., 2003a, "Schema-driven customization of Web services" (demo). In: *Proceedings of the 29th International Conference on Very Large Data Bases (VLDB)*, pp. 1093–1096, Berlim, Alemanha, Setembro.

ABITEBOUL, S., BONIFATI, A., COBENA, G., MANOLESCU, I. e MILO, T., 2003b, "Dynamic XML documents with distribution and replication". In: *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pp. 527–538, San Diego, EUA, Junho.

ABITEBOUL, S., ALEXE, B., BENJELLOUN, O., CAUTIS, B., FUNDULAKI, I., MILO, T. e SAHUGUET, A., 2004a, "An electronic patient record 'on steroids': Distributed, peer-to-peer, secure and privacy-conscious (demo)". In: *Proceedings of the 30th International Conference on Very Large Data Bases (VLDB)*, pp. 1273–1276, Toronto, Canada, Agosto.

ABITEBOUL, S., BENJELLOUN, O., CAUTIS, B., MANOLESCU, I., MILO, T. e PREDA, N., 2004b, "Lazy Query Evaluation for Active XML". In: *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pp. 227–238, Paris, França, Junho.

ABITEBOUL, S., BENJELLOUN, O., MANOLESCU, I., MILO, T. e WEBER, R., 2004c, "Active XML: A Data-Centric Perspective on Web Services". In: Levene, M., Poulovassilis, A. (eds), *Web Dynamics – Adapting to Change in Content; Size, Topology and Use*, ISBN 3-540-40676-X, capítulo 15, Springer.

ABITEBOUL, S., BENJELLOUN, O. e MILO, T., 2004d, "Positive Active XML". In: *Proceedings of the 23rd ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS)*, pp. 35–45, Paris, França, Junho.

ABITEBOUL, S., MANOLESCU, I. e PREDA, N., 2004e, "Constructing and querying a peer-to-peer warehouse of XML resources". In: *Proceedings of the 2nd International Workshop on Semantic Web and Databases*, pp. 219–225, Toronto, Canada, Agosto.

ABITEBOUL, S., ABRAMS, Z., HAAR, S. e MILO, T., 2005a, "Diagnosis of Asynchronous Discrete Event Systems: Datalog to the Rescue!". In: *Proceedings of the 24th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS)*, pp. 358–367, Baltimore, EUA, Junho.

ABITEBOUL, S., AGRAWAL, R., BERNSTEIN, P.A., CAREY, M.J., CERI, S., CROFT, W.B., DEWITT, D.J., FRANKLIN, M.J., GARCIA-MOLINA, H., GAWLICK, D., GRAY, J., HAAS, L.M., HALEVY, A.Y., HELLERSTEIN, J.M., IOANNIDIS, Y.E., KERSTEN, M.L., PAZZANI, M.J., LESK, M., MAIER, D., NAUGHTON, J.F., SCHEK, H.-J., SELLIS, T.K., SILBERSCHATZ, A., STONEBRAKER, M., SNODGRASS, R.T., ULLMAN, J.D., WEIKUM, G., WIDOM, J., ZDONIK, S., 2005b, "The Lowell database research self-assessment", *Communications of the ACM (CACM)*, v. 48, n. 5 (Maio), pp. 111–118.

ABITEBOUL, S., BENJELLOUN, O., MANOLESCU, I., MILO, T., 2005c. *The Active XML project: an overview*, Relatório Técnico (Gemo *Report*) 331, Institut National de Recherche en Informatique et en Automatique (INRIA), Outubro.

ABITEBOUL, S., LEROY, X., VRDOLJAK, B., BRYCE, C., DICOSMO, R., DITTRICH, K., FERMIGIER, S., MILO, T., SAGI, A., SHTOSSEL, Y., LAURIERE, S., LEPIED, F., POP, R., VILLARD, F., PANTO, E. e SMETS, J.-P., 2005d, "EDOS: Environment for the Development and Distribution of Open Source Software". In: *Proceedings of the 1st International Conference on Open Source Software Systems (OSS)*, anais eletrônicos (http://oss2005.case.unibz.it/Papers/37.pdf), Gênova, Itália, Julho.

ABITEBOUL, S., MANOLESCU, I. e TAROPA, E, 2006a, "A framework for distributed XML data management". In: *Proceedings of the 10th International Conference on Extending Database Technology (EDBT)*, pp. 1049–1058, Munique, Alemanha, Março.

ABITEBOUL, S., NGUYEN, B. e RUBERG, G., 2006b, "Building an Active Content Warehouse", In: Darmont, J., Boussaïd, O. (eds), *Processing and Managing Complex Data for Decision Support*, ISBN 1-59140-655-2, capítulo III, Hershey, EUA, Idea Group Publishing.

ABRAMS, Z. e LIU, J., 2006, "Greedy is Good: On Service Tree Placement for In-Network Stream Processing". In: *Proceedings of the 26th IEEE International Conference on Distributed Computing Systems (ICDCS)*, pp. 72, Washington, EUA, Julho.

ACTIVEXML, 2002, *Active XML*, disponível em: http://www.activexml.net (último acesso: 07/11/2006). Código livre publicado pelo *ObjectWeb*, em: http://forge.objectweb.org/projects/activexml/ (último acesso: 07/11/2006).

ACTIVEXML TEAM, 2005, *Active XML User's Guide*, disponível em: http://activexml.net/AXML%20Guide.pdf (último acesso: 07/11/2006).

AGUILERA, V., 2002, *X-OQL Homepage*, disponível em: http://activexml.net/xoql (último acesso: 08/01/2007).

AHO, A., HOPCROFT, J. e ULLMAN, J., 1983, *Data Structures and Algorithms*. 1 ed., Massachusetts, Addison-Wesley.

ALHUSAINI, A.H., PRASANNA, V.K. e RAGHAVENDRA, C.S., 1999, "A Unified Resource Scheduling Framework for Heterogeneous Computing Environments". In: *Proceedings of the 8th IEEE Heterogeneous Computing Workshop (HCW)*, pp. 156–165, San Juan, Porto Rico, Abril.

ALLEN, G., DAVIS, K., GOODALE, T., HUTANU, A., KAISER, H., KIELMANN, T., MERZKY, A., NIEUWPOORT, R.V., REINEFELD, A., SCHINTKE, F., SCHÜTT, T., SEIDEL, E. e ULLMER, B., 2005, "The Grid Application Toolkit: Towards Generic and Easy Application Programming Interfaces for the Grid", *Proceedings of the IEEE*, v. 93, n. 3, pp. 534–550.

ALONSO, G., GÜNTHÖR, R., KAMATH, M., AGRAWAL, D., ABBADI, A.E. e MOHAN, C., 1995a, "Exotica/FMDC: Handling Disconnected Clients in a Workflow Management System". In: *Proceedings of the 3rd International Conference on Cooperative Information Systems (CoopIS)*, pp. 99–110, Viena, Áustria, Maio.

ALONSO, G., MOHAN, C., GÜNTHÖR, R., AGRAWAL, D., ABBADI, A.E. e KAMATH, M., 1995b, "Exotica/FMQM: A Persistent Message-Based Architecture for Distributed Workflow Management". In: *Proceedings of the IFIP Working Conference on Information Systems for Decentralized Organizations*, pp. 1–18, Trondheim, Noruega, Agosto.

ALPDEMIR, M., MUKHERJEE, A., GOUNARIS, A., PATON, N., FERNANDES, A. e FITZGERALD, D., 2004, "OGSA-DQP: A Service for Distributed Querying on the Grid" (demo). In: *Advances in Database Technology - EDBT, 9th International Conference on Extending Database Technology*, pp. 858–861, Creta, Grécia, Março.

ALPDEMIR, M., MUKHERJEE, A., PATON, N., WATSON, P., FERNANDES, A., GOUNARIS, A. e SMITH, J., 2003, "Service-Based Distributed Querying on the Grid". In: *Proceedings of the 1st International Conference on Service-Oriented Computing (ICSOC)*, pp. 467–482, Trento, Itália, Dezembro.

ANDRADE, A., 2006, *PartiX: Projeto de Fragmentação de Dados XML*. Dissertação de M.Sc., COPPE/UFRJ, Rio de Janeiro, RJ, Brasil.

ANDRADE, A., RUBERG, G., BAIÃO, F., BRAGANHOLO, F. e MATTOSO, M., 2005, *PartiX: processing XQuery queries over fragmented XML repositories*, Relatório Técnico ES-691/05, COPPE, Universidade Federal do Rio de Janeiro, Dezembro.

ANDRADE, A., RUBERG, G., BAIÃO, F., BRAGANHOLO, F. e MATTOSO, M., 2006, "Efficiently processing XML queries over fragmented repositories with PartiX". In: *Proceedings of the 2nd International Workshop on Database Technologies for Handling XML Information on the Web (DataX)*, realizado em conjunto com EDBT 2006, LNCS 4254, pp. 14–25, Munique, Alemanha, Março.

ANTONIU, G., BERTIER, M., BOUGE, L., CARON, E., DESPREZ, F., JAN, M., MONNET, S. e SENS, P., 2005, *GDS: an Architecture Proposal for a Grid Data-Sharing Service*, NUM 5593, Projet GRAAL, Institut National de Recherche en Informatique et en Automatique (INRIA), Montbonnot Saint Ismier, França, Junho.

ARORA, M., DAS, S.K. e BISWAS, R., 2002, "A De-Centralized Scheduling and Load Balancing Algorithm for Heterogeneous Grid Environments". In: *Proceedings of the International Conference on Parallel Processing Workshops (ICPPW)*, pp. 499–505, Vancouver, Canadá, Agosto.

AXIS, 2000, *Apache <Web Services /> Project – Axis*, disponível em: http://ws.apache.org/axis (último acesso: 05/12/2006).

AZEVEDO, V., 2003, *WebTransact-EM: Um Modelo para a Execução Dinâmica de Serviços Web Semanticamente Equivalentes*. Dissertação de M.Sc., COPPE/UFRJ, Rio de Janeiro, RJ, Brasil.

AZIS, N., RUBERG, G., NGUYEN, B. e ABITEBOUL, S., 2004, *Acware Viewer: A reporting tool for Active Content Warehouses*, Relatório de Pesquisa, Institut National de Recherche en Informatique et en Automatique (INRIA-Futurs), Julho.

BABU, S. e BIZARRO, P., 2005, "Adaptive Query Processing in the Looking Glass". In: *Proceedings of the 2nd Biennial Conference on Innovative Data Systems Research (CIDR)*, pp. 238–249, Asilomar, EUA, Janeiro.

BACEN, 2006, *Glossário Completo: Swap*, Banco Central do Brasil, disponível em: http://www.bcb.gov.br/glossario.asp?id=GLOSSARIO&q=swap (último acesso: 03/12/2006).

BARBOSA, D., MENDELZON, A., KEENLEYSIDE, J. e LYONS, K., 2002, "Toxgene: An extensible template-based data generator for XML". In: *Proceedings of the Fifth International Workshop on the Web and Databases, WebDB*, pp. 49-54, Madison, EUA, Junho.

BARROS, A., DUMAS, M. e HOFSTEDE, A., 2005, *Service Interaction Patterns: Towards a Reference Framework for Service-based Business Process Interconnection*, FIT-TR-2005-02, Faculty of Information Technology, Queensland University of Technology, Brisbane, Austrália, Março.

BAUER, T. e DADAM, P., 1997, "A Distributed Execution Environment for Large-Scale Workflow Management Systems with Subnets and Server Migration". In: *Proceedings of the 2nd IFCIS Conference on Cooperative Information Systems (CoopIS)*, pp. 99–108, Kiawah Island, EUA, Junho.

BAUER, T. e DADAM, P., 2000, "Efficient Distributed Workflow Management Based on Variable Server Assignments". In: *Proceedings of the 12th International Conference on Advanced Information Systems Engineering (CAiSE)*, pp. 94–109, Estocolmo, Suécia, Junho.

BENATALLAH, B., DUMAS, M. e SHENG. Q.Z., 2005, "Facilitating the Rapid Development and Scalable Orchestration of Composite Web Services", *Distributed and Parallel Databases*, v. 17, n. 1 (Janeiro), pp. 5–37.

BENJELLOUN, O., 2004, *Active XML: A data-centric perspective on Web services, Thèse d'Informatique (Docteur)*, Université Paris XI, Orsay, França.

BISWAS, D. e KIM, I.-G., 2007, "Atomocity for P2P based XML Repositories". In: *Proceedings of the 2nd International Workshop on Services Engineering (SEIW)*, aceito para publicação, Istambul, Turquia, Abril.

BLYTHE, J., DEELMAN, E. e GIL, Y., 2004, "Automatically composed workflows for grid environments", *IEEE Intelligent Systems*, v. 19, n. 4 (Julho/Agosto), pp. 16–23.

BLYTHE, J., DEELMAN, E., GIL, Y., KESSELMAN, C., AGARWAL, A., MEHTA, G. e VAHI, K., 2003, "The Role of Planning in Grid Computing". In: *Proceedings of the 13rd International Conference on Automated Planning and Scheduling (ICAPS)*, pp. 153–163, Trento, Itália, Junho.

BLYTHE, J, JAIN, S., DEELMAN, E., MANDAL, A. e KENNEDY, K., 2005, "Task Scheduling Strategies for Workflow-based Applications in Grids". In: *Proceedings of the 5th International Symposium on Cluster Computing and the Grid (CCGrid)*, pp. 759–767, Cardiff, Reino Unido, Maio.

BOUGANIM, L., FABRET, F., MOHAN, C. e VALDURIEZ, P., 2000, "Dynamic Query Scheduling in Data Integration Systems". In: *Proceedings of the 16th International Conference on Data Engineering (ICDE)*, pp. 425–434, San Diego, EUA, Fevereiro.

BPMN, 2004, *Business Process Modeling Notation (BPMN) Information*, Object Management Group/Business Process Management Initiative, disponível em: http://www.bpmn.org/ (último acesso: 08/12/2006).

BRADLEY, W. e MAHER, D., 2004, "The NEMO P2P service orchestration framework". In: *Proceedings of the 37th Hawaii International Conference on System Sciences (HICSS)*, pp. 90290c, Big Island, EUA, Janeiro.

BRAGA, B., 2005, *Um Esquema de Votação Dinâmica Descentralizado e Tolerante a Falhas para Redes Peer-to-Peer*. Dissertação de M.Sc., COPPE/UFRJ, Rio de Janeiro, RJ, Brasil.

BRAUN, T., SIEGEL, H., BECK, N., BÖLÖNI, L., MAHESWARAN, M., REUTHER, A., ROBERTSON, J., THEYS, M. e YAO, B., 1998, "A Taxonomy for Describing Matching and Scheduling Heuristics for Mixed-Machine Heterogeneous Computing Systems". In: *Proceedings of the 17th IEEE Symposium on Reliable Distributed Systems*, pp. 330–335, West Lafayette, EUA, Outubro.

BRAUN, T., SIEGEL, H., BECK, N., BÖLÖNI, L., MAHESWARAN, M., REUTHER, A., ROBERTSON, J., THEYS, M., YAO, B., HENSGEN, D. e FREUND, R., 2001, "A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems", *Journal of Parallel and Distributed Computing*, v. 61, n. 6(Junho), pp. 810–837.

BURSTEIN, M., HOBBS, J., LASSILA, O., MARTIN, D., MCDERMOTT, D., MCILRAITH, S., NARAYANAN, S., PAOLUCCI, M., PAYNE, T. e SYCARA, K., 2002, "DAML-S: Web service description for the Semantic Web". In: *Proceedings of the First International Semantic Web Conference*, pp. 348–363, Sardenha, Itália, Junho.

BUSH, V., 1945, "As We May Think", *The Atlantic Monthly*, v. 176, n. 1 (Julho), pp. 101–108.

CANAUD, E., BENBERNOU, S., e HACID, M.-S., 2004, "Managing Trust in Active XML". In: *Proceedings of the 3rd IEEE International Conference on Services Computing (SCC)*, pp. 41–48, Shanghai, China, Setembro.

CAO, J., JARVIS, S.A., SAINI, S. e NUDD, G.R., 2003, "GridFlow: Workflow Management for Grid Computing". In: *Proceedings of the 3rd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID)*, pp. 198–205, Tóquio, Japão, Maio.

CAO, J., JARVIS, S.A., SAINI, S. e NUDD, G.R., 2002, "ARMS: An Agent-based Resource Management System for Grid Computing", *Scientific Programming – Special Issue on Grid Computing*, v. 10, n. 2, pp. 135–148.

CASATI, F., CERI, S., PERNICI, B. e POZZI, G., 1996, "Workflow Evolution". In: *Proceedings of the 15th International Conference on Conceptual Modeling (ER)*, pp. 438–455, Cottbus, Alemanha, Outubro.

CHERKASOVA, L., FU, Y., TANG, W. e VAHDAT, A., 2003, "Measuring and characterizing end-to-end Internet service performance", *ACM Transactions on Internet Technology* (TOIT), v. 3, n. 4 (Novembro), pp. 347–391.

CHEUNG, W., LIU, J., TSANG, K. e WONG, R., 2004, "Towards autonomous service composition in a grid environment". In: *Proceedings of the IEEE International Conference on Web Services (ICWS)*, pp. 550–557, San Diego, EUA, Junho.

CHUN, O., AALST, W.M.P., DUMAS, M. e HOFSTEDE, A.H.M., 2006, *From Business Process Models to Process-Oriented Software Systems: The BPMN to BPEL way*, QUTePrints ID 5266, Queen's University of Technology. Disponível em: http://eprints.qut.edu.au/archive/00005266/01/5266.pdf (último acesso: 08/12/2006).

COLDFUSION, 2006, *ColdFusion MX*, Adobe, disponível em: http://www.adobe.com/products/coldfusion/ (último acesso: 04/10/2006).

COLE, R. e GRAEFE, G., 1994, "Optimization of Dynamic Query Evaluation Plans". In: *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pp. 150–160, Mineápolis, EUA, Maio.

COOPER, B.F. e GARCIA-MOLINA, H., 2005, "Ad-hoc, self-supervising peer-to-peer search networks", *ACM Transactions on Information Systems*, v. 23, n. 2 (Abril), pp. 169–200.

COSTA, L., PIRES, P. e MATTOSO, M., 2004, "Automatic composition of Web services with contingency plans". In: *Proceedings of the IEEE International Conference on Web Services (ICWS)*, pp. 454–461, San Diego, EUA, Junho.

CRESPO, A. e GARCIA-MOLINA, H., 2002, "Routing indices for peer-to-peer systems". In: *Proceedings of the 22nd International Conference on Distributed Computing Systems (ICDCS)*, pp. 23–34, Viena, Áustria, Julho.

CRESPO, A. e GARCIA-MOLINA, H., 2004, "Semantic Overlay Networks for P2P Systems". In: *Proceedings of the Third International Workshop on Agents and Peer-to-Peer Computing (AP2PC)*, pp. 1-13, New York, EUA, Julho.

DAGMAN, 2006, *DAGMan (Directed Acyclic Graph Manager) meta-scheduler for Condor*, Condor Team, University of Wisconsin-Madison, disponível em: http://www.cs.wisc.edu/condor/dagman/ (último acesso: 12/11/2006).

DASILVA, D.P., CIRNE, W. e BRASILEIRO, F.V., 2003, "Trading Cycles for Information: Using Replication to Schedule Bag-of-Tasks Applications on Computacional Grids". In: *Proceedings of the 9th International Euro-Par Conference*, LNCS 2790, pp. 169–180, Klagenfurt, Áustria, Agosto.

DEELMAN, E., BLYTHE, J., GIL, Y., KESSELMAN, C., MEHTA, G., VAHI, K., BLACKBURN, K. LAZZARINI, A., ARBREE, A., CAVANAUGH, R. e KORANDA, S., 2003, "Mapping Abstract Complex Workflows onto Grid Environments", *Journal of Grid Computing*, v. 1, n. 1, pp. 25–39.

DEELMAN, E., BLYTHE, J., GIL, Y., KESSELMAN, C., MEHTA, G., PATIL, S., SU, M.-H., VAHI, K. e LIVNY, M., 2004, "Pegasus: Mapping Scientific Workflows onto the Grid". In: *Proceedings of the 2nd European Across Grids Conference (AxGrids)*, LNCS 3165, pp. 11–20, Nicosia, Chipre, Janeiro.

DEELMAN, E., SINGH, G., SU, M.-H., BLYTHE, J., GIL, Y., KESSELMAN, C., MEHTA, G., VAHI, K., BERRIMAN, B., GOOD, J., LAITY, A., JACOB, J.C. e KATZ, D.S., 2005, "Pegasus: A Framework for mapping complex scientific workflows onto distributed systems", *Scientific Programming*, v. 13, pp. 219–237.

DIAS, F.M.O., CASANOVA, M.A. e CARVALHO, M.T.M., 2003, "Workflow Execution in Disconnected Environments". In: *Anais do XVIII Simpósio Brasileiro de Banco de Dados (SBBD)*, pp. 229–239, Manaus, Brasil, Outubro.

DOAN, A. e HALEVY, A., 2002, "Efficiently ordering query plans for data integration". In: *Proceedings of the 18th International Conference on Data Engineering (ICDE)*, pp. 393–402, San José, EUA, Março.

DONG, F. e AKL, S.G., 2006, *Scheduling Algorithms for Grid Computing: State of the Art and Open Problems*, Relatório Técnico No. 2006-504, School of Computing, Queen's University, Janeiro.

DORST, W.V., 2006, *BogoMips mini-Howto*, versão V38 (Março), disponível em: http://www.clifton.nl/index.html?bogomips.html (último acesso: 05/03/2007).

DUAN, R., FAHRINGER, T., PRODAN, R., QIN, J., VILLAZÓN, A. e WIECZOREK, M., 2005, "Real World Workflow Applications in the Askalon Grid Environment". In: *Proceedings of the European Grid Conference (EGC)*, pp. 454–463, Amsterdã, Holanda, Fevereiro.

EDER, J. e SARINGER, M., 2003, "Workflow Evolution: Generation of Hybrid Flows". In: *Proceedings of the 9th International Conference on Object-Oriented Information Systems (OOIS)*, LNCS 2817, pp. 279–283, Genebra, Suíça, Setembro.

EDOS, 2004, *Environment for the development and Distribution of Open Source software*, disponível em: http://www.edos-project.org (último acesso: 28/11/2006).

ELMASRI, R. e NAVATHE, S.B., 1994, *Fundamentals of Database Systems*, 2 ed. Menlo Park, Addison-Wesley Publishing Company.

EROL, K., HENDLER, J. e NAU, D., 1994, *Semantics for hierarchical task-network planning*, CS-TR- 3239, University of Maryland – Institute for Advanced Computer Studies, College Park, EUA.

FEO, T.A. e RESENDE, G.C., 1995, "Greedy Randomized Adaptive Search Procedures", *Journal of Global Optimization*, v. 6, n. 2, pp. 109–133.

FERRAZ, C., BRAGANHOLO, V. e MATTOSO, M., 2006, "Uma abordagem para o armazenamento de documentos XML ativos". In: *Anais do V Workshop de Teses e Dissertações em Bancos de Dados do XXI Simpósio Brasileiro de Banco de Dados (SBBD)*, pp. 38–42, Florianópolis, Brasil, Outubro.

FERRAZ, C., BRAGANHOLO, V. e MATTOSO, M., 2007, "ARAXA: An approach to manage AXML documents". In: *Anais do XXII Simpósio Brasileiro de Banco de Dados (SBBD)*, aceito para publicação, João Pessoa, Brasil, Outubro.

FERREIRA, F., LUCENA, C. e SCHWABE, D., 2003, "A Peer-To-Peer platform based on Semantic Web Services". In: *Proceedings of the Twelfth International World Wide Web Conference - WWW (Posters)*, Budapeste, Hungria, Maio.

FIGUEIREDO, G., 2007, *Processamento de Consultas sobre Bases XML Distribuídas*. Dissertação de M.Sc., COPPE/UFRJ, Rio de Janeiro, RJ, Brasil.

FIGUEIREDO, G., BRAGANHOLO, V. e MATTOSO, M., 2007, "Um Mediador para o Processamento de Consultas sobre Bases XML Distribuídas". In: *Sessão de Demos do XXII Simpósio Brasileiro de Banco de Dados (SBBD)*, aceito para publicação, João Pessoa, Brasil, Outubro.

FOSTER, I., 2005, "Globus Toolkit Version 4: Software for Service-Oriented Systems". In: *Proceedings of the IFIP International Conference on Network and Parallel Computing (NPC)*, LNCS 3779, pp. 2–13, Pequim, China, Novembro/Dezembro.

FOSTER, I. e KESSELMAN, C., 2000, "Computational Grids". In: *Proceedings of the 4th International Conference on Vector and Parallel Processing (VECPAR)*, LNCS 1981, pp. 3–37, Porto, Portugal, Junho.

FOSTER, I., VÖCKLER, J.-S., WILDE, M. e ZHAO, Y., 2002, "Chimera: A Virtual Data System for Representing, Querying, and Automating Data Derivation". In: *Proceedings of the 14th International Conference on Scientific and Statistical Database Management (SSDBM)*, pp. 37–46, Edimburgo, Escócia, Julho.

GANESAN, P. e GARCIA-MOLINA, H., 2005, "Efficient Queries in Peer-to-Peer Systems", *IEEE Data Engineering Bulletin*, v. 28, n. 1 (Março), pp. 48–54.

GEMO, 2002, *Integration of Data and Knowledge Distributed over the Web*, INRIA-Futurs, Orsay, disponível em: http://gemo.futurs.inria.fr (último acesso: 04/11/2006).

GLOBUS, 2006, *The Globus Toolkit Homepage*, University of Chicago, disponível em: http://www.globus.org/toolkit/ (último acesso: 05/11/2006).

GONG, L., 2001, "JXTA: A network programming environment", *IEEE Internet Computing*, v. 5, n. 3 (Junho), pp. 88–95.

GOUNARIS, A., PATON, N., SAKELLARIOU, R. e FERNANDES, A., 2004a, "Adaptive Query Processing and the Grid: Opportunities and Challenges". In: *Proceedings of the 15th International Workshop on Database and Expert Systems Applications (DEXA Workshops)*, pp. 506–510, Zaragoza, Espanha, Agosto/Setembro.

GOUNARIS, A., SAKELLARIOU, R., PATON, N. e FERNANDES, A., 2004b, "Resource scheduling for parallel query processing on computational Grids". In: *Proceedings of the 5th International Workshop on Grid Computing (GRID)*, pp. 396–401, Pittsburgh, EUA, Novembro.

GRIFFITHS, I., 2004, *Inside XAML*, O'Reilly Online Magazine. Disponível em: http://www.ondotnet.com/pub/a/dotnet/2004/01/19/longhorn.html (último acesso: 02/11/2006).

HÄRDER, T., HAUSTEIN, M.P., MATHIS, C. e WAGNER, M., 2007, "Node labeling schemes for dynamic XML documents reconsidered", *Data & Knowlege Engineering*, v. 60, n. 1, pp. 126–149.

HALEVY, A.Y., IVES, Z.G., SUCIU, D. e TATARINOV, I., 2003, "Schema Mediation in Peer Data Management Systems". In: *Proceedings of the 19th International Conference on Data Engineering (ICDE)*, pp. 505–518, Bangalore, Índia, Março.

HAN, Y., SHETH, A. e BUSSLER, C., 1998, "A Taxonomy for Adaptive Workflow Management". In: *Proceedings of the ACM CSCW-98 Workshop Towards Adaptive Workflow Systems*, disponível em http://lsdis.cs.uga.edu/library/download/HSB98.html (último acesso: 12/09/2005), Seattle, EUA, Novembro.

HÄRDER, T., HAUNSTEIN, M.P., MATHIS, C. e WAGNER, M., 2007, "Node labeling schemes for dynamic XML documents reconsidered", *Data & Knowledge Engineering*, v. 60, n. 1 (Janeiro), pp. 126–149.

HELLERSTEIN, J., FRANKLIN, M., CHANDRASEKARAN, S., DESHPANDE, A., HILDRUN, K., MADDEN, S., RAMAN, V. e SHAH, M., 2000, "Adaptive query processing: Technology in evolution", *IEEE Data Engineering. Bulletin*, v. 23, n. 2 (Junho), pp. 7–18.

HOLLINGSWORTH, D., 1995, *The Workflow Reference Model*, Workflow Management Coalition TC00-1003, Winchester, Reino Unido, Janeiro.

HUANG, L., WALKER, D., HUANG, Y. e RANA, O., 2005 "Dynamic Web service selection for workflow optimization". In: *Proceedings of 4th UK e-Science Programme All Hands Meeting (AHM)*, Nottinghan, Reino Unido, Setembro.

HUEBSCH, R., CHUN, B.N., HELLERSTEIN, J.M., LOO, B.T., MANIATIS, P., ROSCOE, T., SHENKER, S., STOICA, I. e YUMEREFENDI, A.R., 2005, "The Architecture of PIER: an Internet-Scale Query Processor". In: *Proceedings of the Second Biennial Conference on Innovative Data Systems Research*, pp. 28–43, Asilomar, EUA, Janeiro.

HULL, R., LLIRBAT, F., KUMAR, B., ZHOU, G., DONG, G. e SU, J., 2000, "Optimization techniques for data-intensive decision flows". In: *Proceedings of the 16th International Conference on Data Engineering (ICDE)*, pp. 281–292, San Diego, EUA, Fevereiro.

HULL, R. e SU, J., 2005, "Tools for Composite Web Services: A Short Overview", *SIGMOD Record*, v. 34, n. 2 (Junho), pp. 86–95.

IOANNIDIS, Y., NG, R.T., SHIM, K. e SELLIS, T., 1992, "Parametric Query Optimization". In: *Proceedings of the 18th VLDB Conference*, pp. 103–114, Vancouver, Canadá, Agosto.

JANG, S.-H., TAYLOR, V., WU, X., PRAJUGO, M., DEELMAN, E., MEHTA, G. e VAHI, K., 2005, "Prediction-based versus Load-based Site Selection: Quantifying the Difference". In: *Proceedings of the ISCA 18th International Conference on Parallel and Distributed Computing Systems (ISCA PDCS)*, pp. 148–153, Las Vegas, EUA, Setembro.

JDSL, 2000, *The data structures library in Java*, Brown University, disponível em: http://www.cs.brown.edu/cgc/jdsl/ (ultimo acesso: 12/11/2006).

JELLY, 2006, *Jelly: Executable XML*, The Jakarta Project, Apache Software Foundation, disponível em: http://jakarta.apache.org/commons/jelly/ (último acesso: 02/11/2006).

JIM, T. e SUCIU, T., 2001, "Dynamically Distributed Query Evaluation". In: *Proceedings of the Twentieth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS)*, pp. 413–424, Santa Barbara, EUA, Maio.

JXTA, 2006, *Project JXTA*, disponível em: http://www.jxta.org (último acesso: 14/01/2007).

KAMMER, P.J., BOLCER, G.A., TAYLOR, R.N., HITOMI, A.S. e BERGMAN, M., 2000, "Techniques for Supporting Dynamic and Adaptive Workflow", *Computer Supported Cooperative Work*, v. 9, n. 3/4 (Novembro), pp. 269–292.

KEE, Y.-S., LOGOTHETIS, D., HUANG, R., CASANOVA, H. E CHIEN, A., 2005, "Efficient Resource Description and High Quality Selection for Virtual Grids". In: *Proceedings of the 5th IEEE International Symposium on Cluster Computing and the Grid (CCGrid)*, pp. 598–606, Cardiff, Reino Unido, Maio.

KRADOLFER, M. e GEPPERT, A., 1999, "Dynamic Workflow Schema Evolution Based on Workflow Type Versioning and Workflow Migration". In: *Proceedings of the 4th IFCIS International Conference on Cooperative Information Systems (CoopIS)*, pp. 104–114, Edimburgo, Escócia, Setembro.

KREGER, H., 2001, *Web Services Conceptual Architecture (WSCA 1.0)*, White Paper, IBM *Software Group*, Somers, NY, EUA, Maio.

KWOK, Y. e AHMAD, I., 1999, "Static scheduling algorithms for allocating directed task graphs to multiprocessors", *ACM Computing Surveys*, v. 31, n. 4 (Dezembro), pp.406–471.

LEMOS, M. e CASANOVA, M.A., 2006, "On the Complexity of Process Pipeline Scheduling". In: *Anais do Simpósio Brasileiro de Banco de Dados (SBBD)*, pp. 57–71, Florianópolis, Brasil, Outubro.

LEYMANN, F., 2001, *Web Services Flow Language (WSFL 1.0)*, White Paper, IBM *Software Group*, Somers, NY, EUA, Maio.

LIU, Y., NGU, A.H. e ZENG, L., 2004a, "QoS Computation and Policing in Dynamic Web Service Selection". In: *Proceedings of the 13th International Conference on World Wide Web (WWW – Alternate Track Paper & Posters)*, pp. 66–73, Nova York, EUA, Maio.

LIU, L., PU, C. e RUIZ, D., 2004b, "A systematic approach to flexible specification, composition, and restructuring of workflow activities", *Journal of Database Management*, v. 15, n. 1 (Janeiro), pp.1–40.

LUDWIG, H., KELLER, A., DAN, A., KING, R. e FRANCK, R., 2003, *Web Service Level Agreement (WSLA) Language Specification*, Relatório Técnico, IBM Software Group, Janeiro.

MAJITHIA, S., SHIELDS, M., TAYLOR, I. e WANG, I., 2004, "Triana: A Graphical Web Service Composition and Execution Toolkit". In: *Proceedings of the IEEE International Conference on Web Services (ICWS)*, pp. 514–524, San Diego, EUA, Junho.

MAXILIMIEN, E.M. e SINGH, M.P., 2004, "A Framework and Ontology for Dynamic Web Services Selection", *IEEE Internet Computing*, v. 8, n. 5 (Setembro/Outubro), pp. 84-93.

MEDJAHED, B. e BOUGUETTAYA, A., 2005, "A dynamic foundational architecture for semantic Web services", *Distributed and Parallel Databases*, v. 17, n. 2 (Março), pp.179–206.

MENDONÇA, N. e SILVA, J., 2005, "An empirical evaluation of client-side server selection policies for accessing replicated Web services". In: *Proceedings of the 2005 ACM Symposium on Applied Computing (SAC)*, pp: 1704–1708, Santa Fé, EUA, Março.

MEYER, L., 2006, *Estrategias para o Escalonamento Dinâmico de Workflows em Grid*. D.Sc., COPPE/UFRJ, Rio de Janeiro, RJ, Brasil.

MEYER, L.A.V.C., ANNIS, J., WILDE, M., MATTOSO, M. e FOSTER, I., 2006a, "Planning spatial workflows to optimize grid performance". In: *Proceedings of the 2006 ACM Symposium on Applied Computing (SAC)*, pp. 786–790, Dijon, França, Abril.

MEYER, L.A.V.C., SHEFTNER, D., VOECKLER, J., MATTOSO, M. , WILDE, M. e FOSTER, I., 2006b, "An Opportunistic Algorithm for Scheduling Workflows on Grids". In: *Proceedings of the 7th International Meeting on High Performance Computing for Computational Science (VECPAR)*, disponível em http://vecpar.fe.up.pt/2006/programme/papers/15.pdf (último acesso: 09/11/2006), Rio de Janeiro, Brasil, Julho.

MIGNET, L., BARBOSA, D. e VELTRI, P., 2003, "The XML Web: a First Study". In: *Proceedings of the 12th International World Wide Web Conference (WWW)*, pp. 500–510, Budapeste, Hungria, Maio.

MILNER, R., PARROW, J. e WALKER, D., 1992a, "A Calculus of Mobile Processes Pt.1", *Information and Computation*, v. 100, n. 1 (Setembro), pp. 1–40.

MILNER, R., PARROW, J. e WALKER, D., 1992b, "A Calculus of Mobile Processes Pt.2", *Information and Computation*, v. 100, n. 1 (Setembro), pp. 41–77.

MILNER, R., 1993, "The Polyadic Pi-Calculus: A Tutorial". In: Hamer, F.L., Brauer, W. e Schwichtenberg, H. (eds), *Logics and Algebra of Specification*, pp. 203–246, Springer.

MILO, T., ABITEBOUL, S., AMANN; B., BENJELLOUN, O. e DANG-NGOC, F., 2003, "Exchanging Intensional XML Data", In: *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pp. 289–300, San Diego, EUA, Junho.

MOHAN, C., ALONSO, G., GÜNTHÖR, R. e KAMATH, M., 1995, "Exotica: A Research Perspective ob Workflow Management Systems", *IEEE Data Engineering Bulletin*, v. 18, n. 1 (Março), pp. 19–26.

MUTH, P., WODTKE, D., WEISSENFELS, J., KOTZ-DITTRICH, A. e WEIKUM, G., 1998, "From Centralized Workflow Specification to Distributed Workflow Execution", *Journal of Intelligent Information Systems*, v. 10, n. 2 (Março/Abril), pp. 159–184.

NEJDL, W., WOLPERS, M., SIBERSKI, W., SCHMITZ, C. SCLOSSER, T., BRUNKHORST, I. e LÖSER, 2003, "Super-peer-based routing and clustering strategies for RDF-based peer-to-peer networks". In: *Proceedings of the Twelfth International World Wide Web Conference, (WWW)*, pp. 536–543, Budapeste, Hungria, Maio.

NG, W.S., OOI, B.C., TAN, K.-L. e ZHOU, Y., 2003, "PeerDB: A P2P-based System for Distributed Data Sharing". In: *Proceedings of the 19th International Conference on Data Engineering (ICDE)*, pp. 633–644, Bangalore, Índia, Março.

NUDD, G.R., KERBYSON, D.J., PAPAEFSTATHIOU, E., PERRY, S.C., HARPER, J.S. e WILCOX, D.V., 2000, "PACE – A toolset for the performance prediction of parallel and distributed systems", *The International Journal of High Performance Computing Applications*, v. 14, n. 3 (Setembro/Outubro), pp. 228-251.

OASIS, 2007, *Web Services Business Process Execution Language Version 2.0*, Committee Draft, OASIS, Janeiro. Disponível em: http://www.oasis-open.org/committees/download.php/22036/wsbpel-specification-draft_candidate_CD_Jan_25_07.pdf (último acesso: 12/02/2007).

OINN, T., ADDIS, M., FERRIS, J., MARVIN, D., SENGER, M., GREENWOOD, M., CARVER, T., GLOVER, K., POCOCK, M.R., WIPAT, A. e LI, P., 2004, "Taverna: A tool for the composition and enactment of bioinformatics workflows", *Bioinformatics Journal*, v. 20, n. 17 (Novembro), pp. 3045–3054.

OMG, 2006, *Business Process Modeling Notation Specification 1.0*, Object Management Group, *Final Adopted Specification* (Fevereiro).

O'REILLY, T., 2005, "What Is Web 2.0", *O'Reilly Network*, disponível em: http://www.oreillynet.com/pub/a/oreilly/tim/news/2005/09/30/what-is-web-20.html (último acesso: 24/10/2006).

OWLS, 2004, *OWL-S: Semantic Markup for Web Services*, W3C Member Submission, Novembro. Disponível em: http://www.w3.org/Submission/OWL-S/ (último acesso: 07/11/2006).

PAPADIMOS, V. e MAIER, D., 2002, "Mutant Query Plans", *Information & Software Technology*, v. 44, n. 4 (Março), pp. 197–206.

PAPADIMOS, V., MAIER, D.e TUFTE, K., 2003, "Distributed Query Processing and Catalogs for Peer-to-Peer Systems". In: *Proceedings of the First Biennial Conference on Innovative Data Systems Research (CIDR)*, pp. 5–8, Asilomar, EUA, Janeiro.

PAPARIZOS, S., WU, Y., LAKSHMANAN, L. JAGADISH, H., 2004, "Tree Logical Classes for Efficient Evaluation of XQuery". In: *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pp. 71–82, Paris, França, Junho.

PEGASUS, 2006, *Pegasus – Planning for Execution in Grids*, página disponível em: http://pegasus.isi.edu (último acesso: 12/11/2006).

PEREIRA, D., 2007, *Geração eficiente de planos de materialização para documentos XML ativos*. Dissertação de M.Sc., COPPE/UFRJ, Rio de Janeiro, Brasil.

PEREIRA, D., RUBERG, G. e MATTOSO, M., 2005, "Algoritmos para a Geração Eficiente de Planos de Materialização para Documentos XML Ativos". In: *Workshop de Teses e Dissertações em Banco de Dados*, Uberlândia, Brasil, Outubro.

PEREIRA, D., RUBERG, G. e MATTOSO, M., 2006, "Geração Eficiente de Planos de Materialização para Documentos XML Ativos". In: *Anais do XXI Simpósio Brasileiro de Banco de Dados (SBBD)*, pp. 236–250, Florianópolis, Brasil, Outubro.

PETERSON, J.L., 1977, "Petri Nets", *ACM Computing Surveys(CSUR)*, v. 9, n. 3 (Setembro), pp. 223–252.

PINHEIRO, W., VIVACQUA, A., BARROS, R., MATTOS, A., CIANNI, N., MONTEIRO-JR, P., MARTINO, R., MARQUES, V., XEXEO, G. e SOUZA, J., 2007, "Dynamic Workflow Management for P2P Environments using Agents". In: *Proceedings of the 7th International Conference on Computational Science (ICCS)*, pp. 253–256, Pequim, China, Maio.

PIRES, P., 2002, *WebTransact: A Framework for Specifying and Coordinating Reliable Web Services Compositions*. Tese de D.Sc., COPPE/UFRJ, Rio de Janeiro, RJ, Brasil.

PITOURA, E., ABITEBOUL, S., PFOSER, D., SAMARAS, G. e VAZIRGIANNIS, M., 2003, "DBGlobe: a service-oriented P2P system for global computing", *SIGMOD Record*, v. 32, n. 3 (Setembro), pp. 77–82.

PYTHON, 2007, *Python Programming Language – Official Web Site*, disponível em: http://www.python.org (último acesso: 14/02/2007).

RAJAMONY, R. e ELNOZAHY, M., 2001, "Measuring client-perceived response time on the WWW". In: *Proceedings of the 3rd USENIX Symposium on Internet Technologies and Systems (USITS)*, São Francisco, EUA, Março.

RUBERG, G., 2001, *Um Modelo de Custo para o Processamento de Consultas em Bases de Objetos Distribuídos*. Dissertação de M.Sc., COPPE/UFRJ, Rio de Janeiro, RJ, Brasil.

RUBERG, N., 2002, *DIG: Um Serviço para Prover Custos e Estatísticas para o Processamento Distribuído de Consultas*. Dissertação de M.Sc., COPPE/UFRJ, Rio de Janeiro, RJ, Brasil.

RUBERG, G., BAIAO, F. e MATTOSO, M., 2002a, "Estimating Costs of Path Expression Evaluation in Distributed Object Databases". In: *Proceedings of the 13th International Conference on Database and Expert Systems Applications (DEXA)*, LNCS 2453, pp. 1–8, Provence, França, Setembro.

RUBERG, G. e MATTOSO, M., 2007a, *XCraft: A Dynamic Optimizer for the Materialization of Active XML Documents*, Relatório Técnico ES-709/07, COPPE, Universidade Federal do Rio de Janeiro, Maio.

RUBERG, G. e MATTOSO, M., 2007b, "XCraft: Boosting the Performance of Active XML Materialization". In: *submetido à 11th International Conference on Extending Database Technology (EDBT)*, Nantes, França.

RUBERG, N., RUBERG, G. e MANOLESCU, I., 2004a, "Towards cost-based optimization for data-intensive Web service computations". In: *Anais do XIX Simpósio Brasileiro de Bancos de Dados*, pp. 283–297, Brasília, Brasil, Outubro.

RUBERG, N., RUBERG, G. e MANOLESCU, I., 2004b, *Towards cost-based optimization for data-intensive Web service computations*, Relatório Técnico (*Research Report*) No. 5222, Institut National de Recherche en Informatique et en Automatique (INRIA-Futurs), Junho.

RUBERG, N., RUBERG, G. e MATTOSO, M., 2002b, "DIG: Um Serviço de Custos e Estatísticas para o Processamento Distribuído de Consultas". In: *Anais do XVII Simpósio Brasileiro de Banco de Dados (SBBD)*, pp. 121–135, Gramado, Brasil, Outubro.

RUBERG, N., RUBERG, G. e MATTOSO, M., 2003, "Digging Database Statistics and Costs Parameters for Distributed Query Processing". In: *Proceedings of the OTM Confederated International Conferences (CoopIS/DOA/ODBASE)*, LNCS 2888, pp. 301–318, Sicília, Itália, Novembro.

RUSSELL, N., HOFSTEDE, A.H.M., EDMOND, D. e AALST, W.M.P., 2004a, *Workflow Data Patterns*, FIT-TR-2004-01, Faculty of Information Technology, Queensland University of Technology, Brisbane, Austrália.

RUSSELL, N., HOFSTEDE, A.H.M., EDMOND, D. e AALST, W.M.P., 2004b, *Workflow Resource Patterns*, BETA Working Paper Series, WP 127, Eindhoven University of Technology, Eindhoven, Alemanha.

RUSSELL, S. e NORVIG, P., 2003, *Artificial Intelligence: A Modern Approach*. 2 ed., New Jersey, Prentice Hall.

SAKELLARIOU, R. e ZHAO, H., 2004a, "A hybrid heuristic for DAG scheduling on heterogeneous systems". In: *Proceedings of the 18th International Parallel and Distributed Processing Symposium (IPDPS)*, Santa Fé, EUA, Março.

SAKELLARIOU, R. e ZHAO, H., 2004b, "A low-cost rescheduling policy for efficient mapping of workflows on grid systems", *Scientific Programming*, v. 12, n. 4, pp. 253–262.

SARTIANI, C., MANGHI, P., GHELLI, G. e CONFORTI, G., 2004, "XPeer: A Self-Organizing XML P2P Database System". In: *Current Trends in Database Technology - EDBT Workshops (P2P&DB)*, LNCS 3268, pp. 456–465, Heraklion, Grécia, Março.

SCHULER, C., WEBER, R.., SCHULDT, H. e SCHEK, H.-J., 2004, "Scalable Peer-to-Peer Process Management – The OSIRIS Approach". In: *Proceedings of the IEEE International Conference on Web Services (ICWS)*, pp. 26–34, San Diego, EUA, Junho.

SHANNON, V., 2006, "A 'more revolutionary' Web", *International Herald Tribune*, publicado em 24/05/2006.

SINGH, G., KESSELMAN, C. e DEELMAN, E., 2006, "Optimizing Grid-Based Workflow Execution", *Journal of Grid Computing*, v. 3, n. 3–4 (Setembro), pp. 201–219.

SOAP, 2003, *SOAP Version 1.2 Part 0: Primer*, W3C Recommendation, Junho. Disponível em: http://www.w3.org/TR/soap12-part0/ (último acesso: 02/11/2006).

SOUSA, F. e MACHADO, J., 2006, "Um Mecanismo para Replicação em Sistemas de Banco de Dados XML Nativos". In: *Anais do V Workshop de Teses e Dissertações em Bancos de Dados do XXI Simpósio Brasileiro de Banco de Dados (SBBD)*, pp. 32–37, Florianópolis, Brasil, Outubro.

SOUSA, F., CARNEIRO-FILHO, H.J.A. e MACHADO, J., 2007a, "A New Approach for Replication of XML Data". In: *Proceedings of the 18 th International Conference on Database and Expert Systems Applications (DEXA)*, LNCS 4653, pp. 141–150, Regensburg, Alemanha, Setembro.

SOUSA, F., CARNEIRO-FILHO, H.J.A., ANDRADE, R. e MACHADO, J., 2007b, "RepliX: Um Mecanismo para a Replicação de Dados XML". In: *Anais do XXII Simpósio Brasileiro de Banco de Dados (SBBD)*, aceito para publicação, João Pessoa, Brasil, Outubro.

SPOONER, D.P., CAO, J., TURNNER, J.D., LIM-CHOI-KEUNG, H.N., JARVIS, S.A. e NUDD, G.R., 2002, "Localised Workload Management using Performance Prediction and QoS Contracts". In: *Proceedings of the 18th Annual UK Performance Engineering Workshop (UKPEW)*, pp. 69–81, Glasgow, Reino Unido, Julho.

SRIVASTAVA, U., WIDOM, J., MUNAGALA, K. e MOTWANI, R., 2006, "Query Optimization over Web Services". In: *Proceedings of the 32nd International Conference on Very Large Data Bases (VLDB)*, pp. 355–366, Seul, Coréia, Setembro.

STOICA, I., MORRIS, R., KARGER, D., KAASHOEK, F. e BALAKRISHNAN, H., 2001, "Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications". In: *Proceedings of the ACM SIGCOMM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, pp. 149–160, San Diego, EUA, Agosto.

SUCIU, D., 2002, "Distributed query evaluation on semistructured data", *ACM Transactions on Database Systems (TODS)*, v. 27, n. 1 (Março), pp.1–62.

TAROPA, E., LEE, W.-J. e HAN, T.-D., 2005, "Fast Obtaining Weak Code Mobility Using Active XML". In: *Proceedings of the Korea Computer Congress*, v. 32, pp. 16–18, Phoenix Park, Coréia, Julho.

TATARINOV, I., IVES, Z.G., MADHAVAN, J., HALEVY, A.Y., SUCIU, D., DALVI, N.N., DONG, X., KADIYSKA, Y., MIKLAU, G.e MORK, P., 2003, "The Piazza peer data management project", *SIGMOD Record*, v. 32, n. 3 (Setembro), pp. 47–52.

TAVERNA, 2006, *Taverna project website*, disponível em: http://taverna.sourceforge.net (último acesso: 05/11/2006).

TAYLOR, I., SHIELDS, M., WANG, I. e RANA, O., 2003, "Triana Applications within Grid Computing and Peer to Peer Environments", *Journal of Grid Computing*, v. 1, n. 2, pp. 199–217.

THAIN, D., BENT, J., ARPACI-DUSSEAU, A. C., ARPACI-DUSSEAU, R. H. e LIVNY, M., 2003, "Pipeline and Batch sharing in Grid Workloads". In: *Proceedings of the 12th Conference on High-Performance Distributed Computing* (HPDC), pp. 152–161, Seattle, EUA, Junho.

THAIN, D., TANNENBAUM, T. e LIVNY, M., 2005, "Distributed Computing in Practice: the Condor experience", *Concurrency and Computation: Practice and Experience*, v. 17, n. 2-4 (Fevereiro/Abril), pp. 323–356.

TOMCAT, 1999, *The Apache Tomcat Servlet Container*, disponível em: http://tomcat.apache.org (último acesso: 05/12/2006).

TRIANA, 2003, *The Triana Project*, Cardiff University, disponível em: http://www.trianacode.org (último acesso: 08/11/2006).

TUXEDO, 2001, *BEA Tuxedo – Product Overview*, BEA Systems Inc., disponível em: http://edocs.bea.com/tuxedo/tux80/ovrview/overviea.htm (último acesso: 16/12/2006).

UDDI, 2004, *Introduction to UDDI: Important Features and Functional Concepts*, White Paper, Organization for the Advancement of Structured Information Standards (OASIS UDDI), Outubro.

URHAN, T., FRANKLIN, M. e AMSALEG, L., 1998, "Cost based query scrambling for initial delays". In: *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pp. 130–141, Seattle, EUA, Junho.

VARGAS, P.K., DUTRA, I.C. e GEYER, C., 2004, *Application partitioning and hierarchical application management in grid environments*, ES-657/04, COPPE/UFRJ.

VARGAS, P.K., DUTRA, I.C., NASCIMENTO, V.D., SANTOS, L.A.S., SILVA, L.C., GEYER, C.F.R. e SCHULZE, B., 2005 "Hierarchical Submission in a Grid Environment". In: *Proceedings of the 3rd International Workshop on Middleware for Grid Computing (MGC)*, pp. 1–6, Grenoble, França, Novembro.

VERMA, K., GOMADAM, K., SHETH, A.P., MILLER, J.A. e WU, Z., 2005, *The METEOR-S approach for configuring and executing dynamic Web processes*, LSDIS 05-001, University of Georgia, Athens, EUA.

VIEIRA, H., RUBERG, G. e MATTOSO, M., 2002, "XVerter: Armazenamento e Consulta de Dados XML em SGBDs". In: *Anais do XVII Simpósio Brasileiro de Banco de Dados (SBBD)*, pp. 224–238, Gramado, Brasil, Outubro.

VIEIRA, H., RUBERG, G. e MATTOSO, M., 2003, "XVerter: Querying XML Data with OR-DBMS". In: *Proceedings of the 5th ACM CIKM International Workshop on Web Information and Data Management (WIDM)*, pp. 37–44, New Orleans, EUA, Novembro.

W3C, 2006, *W3C: World Wide Web Consortium*, disponível em: http://www.w3.org (último acesso: 02/11/2006).

WEB SERVICES, 2006, *W3C Web Services Activity*, disponível em: http://www.w3.org/2002/ws (último acesso: 02/11/2006).

WEBSPHERE, 2007, *Providing a messaging backbone for SOA connectivity*, White Paper, IBM *Software Group*, Sommers, NY, Estados Unidos, Março.

WEBSPHERE MQ WORKFLOW, 2005, *IBM WebSphere MQ Workflow: Concepts and Architecture (Version 3.6)*, publicação nº GH12-6285-07, IBM Corp., Estados Unidos, Junho.

WIDOM, J., 1996, "The Starburst Active Database Rule System". In: Widom, J., Ceri, S. (eds), *Active Database Systems: Triggers and Rules for Advanced Database Processing*, pp. 87-109, São Francisco, EUA, Morgan-Kaufmann.

WIECZOREK, M., PRODAN, R. e FAHRINGER, T., 2005, "Scheduling of scientific workflows in the ASKALON grid environment", *SIGMOD Record*, v. 34, n. 3 (Setembro), pp.56–62.

WFMC, 1999, *Workflow Management Coalision Terminology & Glossary*, Documento nº WFMC-TC-1011, disponível em: http://www.wfmc.org/standards/docs/TC-1011_term_glossary_v3.pdf (último acesso: 08/12/2006).

WODTKE, D., WEISSENFELS, J., WEIKUM, G. e KOTZ-DITTRICH, A., 1996, "The Mentor Project: Steps Towards Enterprise-Wide Workflow Management". In: *Proceedings of the 12th International Conference on Data Engineering (ICDE)*, pp. 556–565, New Orleans, EUA, Março.

WSDL, 2006, *Web Services Description Language (WSDL) Version 2.0 Part 0: Primer*, W3C Candidate Recommendation, Março. Disponível em: http://www.w3.org/TR/2006/CR-wsdl20-primer-20060327/ (último acesso: 02/11/2006).

WU, M.-Y., SHU, W. e GU, J., 2001, "Efficient local search for DAG scheduling", *IEEE Transactions on Parallel and Distributed Systems*, v.12, n. 6 (Junho), pp. 617–627.

WU, X., TAYLOR, V. e PARIS, J., 2006, "A Web-based Prophesy Automated Performance Modeling System". In: *Proceedings of the IASTED International Conference on Web Technologies, Applications and Services (WTAS)*, ref. 524–012, Calgary, Canadá, Julho.

XALAN, 1999, *The Apache Xalan Project*, disponível em: http://xalan.apache.org (último acesso: 05/12/2006).

XERCES, 1999, *The Xerces Parser Project*, disponível em: http://xerces.apache.org (último acesso: 05/12/2006).

XMARK, 2003, *The XMark benchmark*, disponível em: http://monetdb.cwi.nl/xml (último acesso: 12/11/2006).

XML, 2006, *Extensible Markup Language (XML) 1.0 (Fourth Edition)*, W3C Recommendation, Agosto. Disponível em: http://www.w3.org/TR/REC-xml (último acesso: 02/11/2006).

XPATH, 1999, *XML Path Language (XPath) Version 1.0*. Disponível em: http://www.w3.org/TR/xpath (último acesso: 12/11/2006).

YU, J. e BUYYA, R., 2004, "A Novel Architecture for Realizing Grid Workflow using Tuple Spaces". In: *Proceedings of the 5th International Workshop on Grid Computing (GRID)*, pp. 119–128, Washington, EUA, Novembro.

YU, J. e BUYYA, R., 2005, "A taxonomy of scientific workflow systems for grid computing", *SIGMOD Record*, v. 34, n. 3 (Setembro), pp. 44–49.

YU, J., BUYYA, R e THAM, C.K., 2005, "Cost-based Scheduling of Scientific Workflow Applications on Utility Grids". In: *Proceedings of the 1st International Conference on e-Science and Grid Technologies (e-Science)*, pp. 140–147, Melbourne, Austrália, Dezembro.

ZENG, L., BENATALLAH, B., NGU, A.H.H., DUMAS, M., KALAGNANAM, J., e CHANG, H., 2004, "QoS-Aware Middleware for Web Services Composition", *IEEE Transactions on Software Engineering*, v. 30, n. 5 (Maio), pp. 311–327.

ZHANG, Y., MANDAL, A., CASANOVA, H., CHIEN, A., KEE, Y.-S., KENNEDY, K. e KOELBEL, C., 2006, "Scalable Grid Application Scheduling via Decoupled Resource Selection and Scheduling". In: *Proceedings of the 6th IEEE International Symposium on Cluster Computing and the Grid (CCGRID)*, pp. 568–575, Cingapura, Maio.

ZHANG, Q., ZOU, Y., TONG, T., MCKEGNEY, R. e HAWKINS, J., 2005, "Automated Workplace Design and Reconfiguration for Evolving Business Processes". In: *Proceedings of the 2005 Conference of the Centre for Advanced Studies on Collaborative Research (CASCON)*, pp. 320–333; Toronto, Canadá, Outubro.

ANEXO I – RELATÓRIO TÉCNICO

A seguir, é apresentado um relatório que detalha os algoritmos e técnicas de otimização propostas no XCraft.

XCraft: A Dynamic Optimizer for the Materialization of Active XML Documents

Gabriela Ruberg, Marta Mattoso
Department of Computer Science – COPPE/UFRJ, Brazil
{gruberg, marta}@cos.ufrj.br

Technical Report ES-709/07
May, 2007

Abstract

An active XML (AXML) document contains special tags that represent calls to Web services. Retrieving its contents consists in materializing *its data elements by invoking all its embedded service calls in a P2P network. In this process, the results of some service calls are often used as inputs to other calls. Also, usually several peers provide each requested Web service, and peers can collaborate to invoke these services. This implies many equivalent materialization alternatives, with different performance.*

Optimizing the AXML materialization process is a hard problem, which often involves searching a huge space of solutions. Current techniques for workflow scheduling and distributed query processing are insufficient for this problem, since in AXML materialization: (i) the set of participating peers is not known in advance; (ii) service calls in the result of other calls forbid a simple "optimize-then-execute" strategy; and (iii) due to the peer volatility in the network, a plan computed by the optimizer may become invalid at the moment of its execution. Moreover, most of the current optimizers are based on centralized coordination.

We propose a dynamic, cost-based optimization strategy *to efficiently materialize AXML documents considering the volatility of a P2P scenario. We formalize the problem from a performance-oriented perspective, and present an optimization strategy that incrementally generates and executes materialization plans. This enables the optimizer to reduce the size of the search space, get more up-to-date information on the status of the peers, and deliver partial results earlier. Our strategy can handle arbitrarily complex AXML documents, and exploits decentralization in many levels.*

We also present a service-oriented optimization architecture called ***XCraft****. We evaluated our approach in an XCraft prototype for the ActiveXML system, an open-source P2P platform. Our results show promising performance gains compared to centralized, static materialization strategies.*

Contents

1 Introduction **188**

2 Running application **189**

3 AXML Basics **191**

4 Service Invocation Constraints in AXML **193**
4.1 Precedence Constraints 193
4.2 Consequence Constraints 194
4.3 Dependency Graph 194
4.4 Dynamic Graph Updates 198

5 Materializing AXML Documents **201**
5.1 Equivalent, Replicated and Generic Services 202
5.2 Exploring P2P Collaboration in AXML . . 203
5.3 Enacting AXML Materialization 204
5.4 The AXML Optimization Problem 205
5.5 Main Problem Topologies 208

6 Optimizing AXML Materialization **209**
6.1 Dynamic Optimization Strategy 209
6.2 Generating Materialization Plans 210
6.3 Determining Materialization Tasks 214
6.4 Dynamic Plan Generation 215
6.5 Delegating AXML Optimization 221

7 Cost Analysis of AXML Materialization **222**
7.1 Representing Heterogenous Scenarios . . . 223
7.2 Heuristic Cost Analysis 223
7.3 Costs of Plan Operators 224

8 XCraft Architecture **224**

9 Experimental Results **226**
9.1 Devising the Search Space 227
9.2 Plan Delegation Effects 228

10 Related Work **228**

11 Conclusions **229**

1 Introduction

Data management techniques for peer-to-peer (P2P) systems have been extensively exploited in the last years. Two technologies have been crucial in this scenario: the XML format, as the universal media for data exchange; and Web services, as the standard for program and data interoperation. Web services are described as independent, self-contained programs whose interfaces can be published, discovered and invoked throughout the Web [60]. They encapsulate heterogeneous business processes, and their related standards [49, 52, 62] are a practical and effective underpinning for reconciling disparate systems. The *combination of XML and Web services* has enabled new models to express powerful distributed computations, raising a new class of *active XML documents* [5]. These documents consist of a highly-adaptive media for distributed information. Hence, solutions to efficiently support them can significantly contribute towards Web computing.

Besides regular XML data, active XML (AXML, for short) documents contain special XML elements which represent calls to Web services. These embedded service calls can be invoked automatically or on-demand; once a service call is invoked, its result is gathered and merged (according to some predefined criteria) into the corresponding XML document. Invoking embedded calls can be thought of as *materializing some intensional data* of the AXML document. Therefore, to retrieve the contents of a document, all of its service calls need to be materialized. Notice this is a quite common scenario in Web applications, where delivering XML documents usually requires materializing their contents first. We are interested in the efficient materialization of AXML documents in a P2P system.

Materializing AXML documents is quite similar to executing workflows: embedded service calls are tasks to be performed, which are often related to each other, causing some invocation constraints and data flows. For example, *invocation dependencies* occur when service calls takes the result of other calls as input parameters. In this case, nested service calls must be invoked first to provide input for their respective outer service calls. These dependencies enforce some precedence constraints on the materialization process, and they often imply some *data flows* between service invocations. Some of these invocation results are required to embody the final materialized document, while others are *intermediate results that do not need to be kept to the end of the materialization*. There may also exist *invocation consequences* in an AXML document, when materializing a service call should automatically trigger another call. Invocation constraints of embedded service calls correspond to some basic control flow patterns, namely sequence, parallel split, and synchronization [54]. However, AXML materialization always involves some data flows towards the peer that is gathering the document contents (called *master peer*). Hence, an AXML document can be incrementally composed and consumed, while partial results are seldom meaningful in workflow systems.

Another issue in materializing AXML documents comes from *intensional answers*. Namely, in AXML-enabled systems, *service calls may return other service calls* as the result of their invocation. This means the problem specification (the actual document to materialize) may evolve *at runtime*. This is very different from traditional distributed query optimization settings: a query can be optimized either statically or dynamically [16, 22], but the query specification itself does not change during optimization (except for parameterized queries). With AXML documents, the system must be able to dynamically update materialization plans accordingly. Also, ideally the system should reduce the scope of impact of these changes in the planning process, thus avoiding excessive reoptimizations.

Basically, the intensional nodes of an AXML document point to specific service references, including the service URL and other parameters that are required to invoke a Web service (as defined in the SOAP and WSDL standards [59]). In a more flexible approach, Web services can be addressed by abstract references, *e.g.*, based in some ontology of services, as in OWL-S [41]. This approach is very convenient to describe AXML data, specially because locating the best resources to execute service requests in a P2P system is often burdensome for users. Considering abstract service references, an AXML document can be materialized by many alternative strategies [44]. Regardless of the services invocation order, these strategies may differ in the choice of: (*i*) the peer that *executes* each service call; and (*ii*) the peer that *invokes* each service call. Observe that possibly several peers can invoke a service call, even if it refers to a specific Web service endpoint. This means that the master peer can delegate the invocation of a Web service to another peer. Such a decentralized approach adheres to a typical P2P execution model, and it can reduce communication costs since it avoids transfering intermediary results to the master peer.

Efficiently materializing an AXML document concerns both determining which peer invokes and/or executes each service call (by taking into account, *e.g.*, their communication costs), and ordering the invocation of relevant service calls (by exploiting possible parallel executions). This problem is complicated by the *membership fluctuations* of a P2P system, where peers can join or leave the community at any time. Therefore, the optimizer cannot afford to spend much effort to generate plans that may be no longer valid at runtime. In fact, unpredictability is endemic in large-scale systems, and *peers are not required to generate complete materialization plans before starting their evaluation*. Instead, partial plans can be generated and executed (possibly in parallel). This approach has several advantages. First, the

optimizer can access *up-to-date knowledge* about the system, which increases both the quality of the plan statistics and the plan validity. Going further, *much of the decision process can be deferred* until the system is better informed on service statistics and/or service location.

Contributions. In this paper, we show that dynamic techniques are vital for efficiently materializing AXML documents. As a consequence, we propose a dynamic optimization strategy that addresses Web service invocations along with the volatility of a heterogeneous P2P environment. We make the following contributions:

- ***A canonical model to represent service invocation constraints of AXML documents.*** We capture relevant issues related to the invocation dependencies and consequences between service calls into a DAG-based representation, and we define the necessary criteria to check its validity (w.r.t. deadlock and execution termination) and to eliminate redundancy. Furthermore, we propose efficient techniques to update the graph of an AXML document at runtime with intensional answers;

- ***A P2P enactment model for AXML materialization with abstract service references.*** We characterize the main participants of the AXML materialization process, and their possible interactions in a P2P scenario. We use this enactment model to define the search space of materialization alternatives for AXML documents, and to formalize the corresponding optimization problem;

- ***A dynamic algorithm to generate and evaluate AXML materialization alternatives.*** We propose an optimization algorithm that can handle arbitrarily complex AXML documents. This algorithm splits the materialization problem into smaller parts, and then interleaves planning and execution, thus enabling the system to yield partial materialization plans and deliver partial results earlier. Plans are encoded with operators of a materialization algebra, which we introduce to properly evaluate embedded service calls;

- ***A cost model to evaluate AXML materialization alternatives.*** We devise a cost-based strategy which represents typical characteristics of P2P systems, such as replicated Web services, heterogeneous machines and communication links, parallel execution, and the delegation of materialization tasks; and

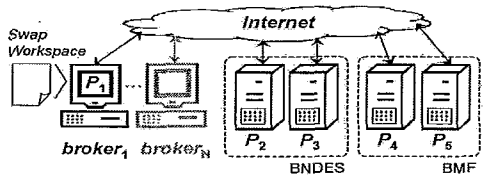
- ***An AXML optimizer architecture and prototype.*** We outline a decentralized, service-oriented optimizer architecture, called **XCraft**, to support the proposed techniques. We implemented an XCraft prototype upon the ActiveXML system [13], an open-source P2P platform to manage AXML documents. We describe an experimental evaluation of its effectiveness.

Although the proposed strategy can benefit AXML materialization in general, it is specially targeted at AXML documents containing calls to *data-intensive Web services*, involving large input or output transfers within a heterogeneous P2P network. Despite the great improvements that we have witnessed in communication speed, data transfers through Web services protocols remain costly due to operations such as packing/unpacking and parsing XML data [44]. Moreover, in such a scenario the main optimization goal is not to find best plans, but mostly to *avoid the worst ones* (which may become unfeasible on critical performance). Also, materialization plans may fail due to many reasons, such as security restrictions and hardware errors. Yet, we believe a dynamic strategy contributes to improve system recovery, since smaller tasks can be better monitored to enable early error detection and fixing.

Paper outline. This paper is organized as follows. We describe in Section 2 an application to motivate the need of optimization for AXML materialization, and in Section 3 we present some basic concepts in AXML documents. We define the canonical formalism to represent service invocation constraints in Section 4. Section 5 describes the P2P enactment model for AXML materialization, and states the optimization problem addressed in this paper. Our optimization strategy is proposed in Section 6, including its dynamic algorithm and the algebra to encode materialization plans. In Section 7, we detail the cost model used to analyze alternative materialization plans. We describe the XCraft optimizer architecture in Section 8. Experimental results obtained with an XCraft prototype are analyzed in Section 9. Section 10 discusses related work, and Section 11 closes with some conclusions and perspectives.

2 Running application

There are many interesting applications for AXML documents [6], such as RSS news syndication [2] and management of eletronic patient records [1]. In the Acware system [10], AXML documents are explored to build *active content warehouses*, which help biologists to continuously integrate and transform information for food risk assessment. Abiteboul *et al.* [9] describe an application based on AXML documents to manage the production and distribution of Open Source software, in the *eDos* European project. In [44], AXML documents are used as a practical framework for a financial application, to support a loan program for farming activities. In this paper, we illustrate the main AXML materialization issues with a financial application for *foreign exchange swap*, named *CurrencySwap*.



(a)

Service	Providers
CheckSwapStatus	P_2, P_3, P_4
GetCurrentSwaps	P_4, P_5
GetSwapLimit	P_2, P_3, P_4
GetContractPrincipal	P_4, P_5
CalculateDebt	P_2, P_3, P_4, P_5
GetContractSwaps	P_4, P_5
GetExchangeRate	P_2, P_3, P_4, P_5
GetLocalDate	P_1, P_2, P_3, P_4, P_5
GetContractPDF	P_4, P_5
ExtractExcerpt	P_1, P_4, P_5

(b)

Figure 1. The *CurrencySwap* application (a) in a P2P setting, and (b) its Web services.

Basically, currency swap operations rely on exchanging debts made in a specific currency against either another foreign currency or a fixed interest rate. For example, suppose a Brazilian company has a contract for a loan in US dollars. As a security against adverse exchange rate movement, this company can negotiate with a *funding bank* to swap its debt currency against a fixed interest rate. This debt is converted into Brazilian *Reais* according to the exchange rate of the swap operation date. On the contract settlement date, if US dollars have become more expensive, then the company will only have to pay the converted debt plus the interest rate, and the funding bank will provide the remaining difference.

An interesting fact about most of the financial applications is that performance is just as important as other tradicionally-critical issues, such as security and reliability. For instance, stock trading systems operate in near-realtime. Hence, optimization is a strong requirement of these systems in distributed settings.

***CurrencySwap* setting.** Figure 1(a) shows the *CurrencySwap* application in a P2P setting. Companies interact with the system through *brokers*. The central player is the Brazilian Mercantile&Future Exchange (BMF), which manages all the swap operations coming from brokers. Swap contracts are negociated with BNDES, the major Brazilian funding bank. In turn, BNDES limits the amount of debt subject to swapping for each company, to reduce its financial risk. In Figure 1(a), dotted lines indicate peers in the same intranet (*e.g.*, peers P_3 and P_4 in the BMF intranet). We assume data transfers in an intranet are 50 times faster than through an Internet connection. Information on

```
<current_contract><number> 12345 </number>
  <company><name>XTechno Acme Ltd</name>
    <can_swap><sc id="1" service="CheckSwapStatus">
      <param name="swaps">
        <sc id="2" service="GetCurrentSwaps">
          <xpath>//company/name</xpath></sc></param>
      <param name="current_limit">
        <sc id="3" service="GetSwapLimit">
          <param name="company">
            <xpath>//company/name</xpath></param>
          <param name="date">
            <xpath>/current_contract/today</xpath></param>
          </sc></param></sc></can_swap></company>
  <principal><sc id="4" service="GetContractPrincipal">
    <xpath>/current_contract/number</xpath></sc></principal>
  <swap_debt><sc id="5" service="CalculateDebt" followed_by="1">
    <param name="principal">
      <xpath>/current_contract/principal/amount</xpath></param>
    <param name="swaps">
      <sc id="6" service="GetContractSwaps">
        <xpath>/current_contract/number</xpath></sc></param>
    <param name="rate"><sc id="7" service="GetExchangeRate">
      <param name="foreign_currency">
        <xpath>/current_contract/principal/currency</xpath>
        </param>
      <param name="date"><xpath>/current_contract/today
        </xpath></param></sc></param>
    </sc></swap_debt>
  <today><sc id="8" service="GetLocalDate"/></today>
  <contract_excerpt><sc id="9" service="ExtractExcerpt">
    <param name="text">
      <sc id="10" service="GetContractPDF"/></param>
    <param name="input_format">PDF</param>
    <param name="output_format">XML</param>
</sc></contract_excerpt></current_contract>
```

(a)

```
<current_contract><number> 12345 </number>
  <company><name>XTechno Acme Ltd</name>
    <can_swap>yes</can_swap></company>
  <principal><amount>75000</amount>
    <currency>USD</currency>
    <due>06/20/2006</due></principal>
  <swap_debt>
    <amount>196500</amount><flow>-15720</flow>
    <currency>BRR</currency></swap_debt>
  <today>04/15/2005</today>
  <contract_excerpt> ... </contract_excerpt>
</current_contract>
```

(b)

Figure 2. *SwapWorkspace* document at P_1, (a) before and (b) after the materialization of sc5.

swap contracts and financial indices are published through Web services; Figure 1(b) lists the main Web services provided by each peer. Peers can gather Web services descriptions either directly from service providers or from catalogs available on the network, such as UDDI servers [52].

During business negotiations, brokers can follow swap information for relevant contracts in a *SwapWorkspace* document, such as the one in Figure 2(a) (in a simplified AXML notation). Basically, the *SwapWorkspace* document contains the contract number, the company name and its swap status at BNDES, the debt principal in foreign currency, the corresponding converted amount (due to swap operations), the current date, and an excerpt of the contract settlement. The contents of the *SwapWorkspace* document must

be gathered from Web services by the invocation of embedded calls, which are represented by the "sc" elements. We denote a service call element as scX, where X is the value of its "id" attribute. In our example, the broker just need to set the contract number and the company name, and then to request (either on-demand or periodically) the system to refresh the workspace contents. A materialized version of the *SwapWorkspace* document is shown in Figure 2(b).

Materializing AXML data. The "sc" elements of the *SwapWorkspace* document refer to Web services that are provided by several different peers (see Figure 1(b)). When a service call is invoked at a peer, the system can lookup for its possible service providers, and then choose the best peer to execute the call. To materialize an entire AXML document, such a decision is usually influenced by the invocation of other service calls in the document, specially when some input parameters contain other "sc" elements. A peer may decide to delegate some related calls to be invoked at another peer, gathering only the results that are necessary to build the AXML document. For example, sc9 takes as input the result of sc10 in Figure 2(a), but only the result of sc9 is required to build *SwapWorkspace*.

Figure 3 illustrates three alternatives to materialize sc9 and sc10. The left-most alternative represents a centralized strategy (P_1 invokes both service calls), whereas in the two others P_1 delegates service invocations to either P_4 (on the center) or P_5 (on the right). Delegation strategies are particularly interesting to evaluate nested service calls when the respective executors can communicate through a link faster than the link to the master peer.

Optimization opportunities. Many different evaluation strategies can be used to materialize the *SwapWorkspace* document, considering all the invocation possibilities of each embedded service call. The materialization performance may vary a lot for each strategy. For example, if transfering the result of sc10 through an Internet connection costs $50s$ (*e.g.*, from P_5 to P_1), then it would cost only $1s$ by intranet transfers. For large data transfers and many service calls, such a difference can be much more important. Thus, a naive materialization strategy may lead to an unacceptable execution time.

On the other hand, optimizing the materialization of AXML documents raises a hard problem: the number of alternatives grows dramatically even for restricted scenarios. For instance, in our simple *CurrencySwap* example, there are at least 1.898×10^{16} alternative materialization plans [1] for the *SwapWorkspace* document. Suppose each plan is generated and analyzed in $0.5ms$ (a quite reasonable measure for modern PCs). An exhaustive search would last more than 305 thousand years (sic!) to find the best plan.

[1] Considering only the possible combinations of peers for service executions and invocations.

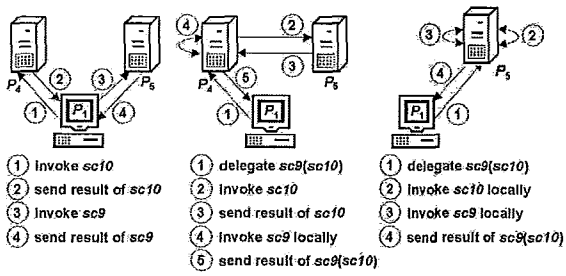


Figure 3. Some alternatives for AXML materialization.

Some heuristics can significantly reduce this search space. In our example, we can apply the *Divide&Conquer* (D&C) heuristic [44], which partitions the document materialization into independent tasks. This would reduce the search space to approximately 1.265×10^{14} alternative plans. Still, this means more than 2 thousand years only to choose the best plan, to actually start materializing the document. Despite the great improvement, the time spent in optimization remains critical. Notice that our example has only 5 distinct peers; if this number raises to 10 peers, the search space becomes 4096 times larger (with the D&C heuristic). In Section 5.4, we provide some formula to estimate the number of AXML materialization alternatives.

Clearly, such optimization delays are not acceptable when materializing an AXML document. Exhaustive and optimal search methods are often unfeasible in AXML settings. Therefore, it is imperative to reduce the search space of materialization alternatives to a manageable size, regardless of the size of the problem. Greedy and opportunistic approaches are typical solutions for complex workflow planning. Nonetheless, they are insufficient for AXML materializarion since they are based on local decisions, which cannot explore delegation to reduce communication costs. In this paper, we present a cost-based strategy to enumerate and rank AXML materialization plans, based on a hibrid metaheuristic. The core of our strategy is a dynamic algorithm that progressively generates and evaluates these plans, thus avoiding to explore complete search spaces, while still considering relevant performance properties.

3 AXML Basics

A typical AXML environment involves a P2P system, whose peers can execute and/or invoke Web services, and an AXML document model that combines XML data and service calls. In this section, we present the system architecture and the document model (borrowed from [3, 5, 39]) defined for the ActiveXML framework [13], an open-source P2P platform that supports AXML documents.

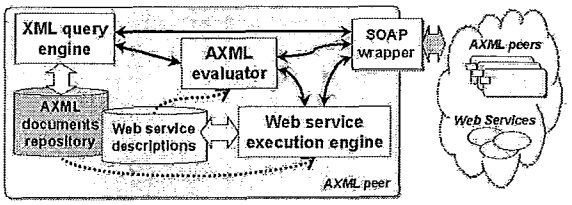


Figure 4. Outline of an ActiveXML peer.

System architecture. The architecture of an ActiveXML peer is depicted in Figure 4. Basically, a peer holds a repository to store local AXML documents, and a catalog of Web service descriptions. The XML query engine provides access to the AXML documents repository, and (when necessary) can request the AXML evaluator to materialize some AXML content. In turn, the AXML evaluator triggers embedded service calls, and updates the corresponding AXML documents accordingly. The invocation of triggered service calls is handled by the Web service execution engine. Observe that each ActiveXML peer can act both: as a *server*, by providing Web services; and as a *client*, when invoking service calls that are embedded in local documents. In particular, an ActiveXML peer can provide a distinguished class of Web services, called *declarative services*, which consist of parameterized queries over local documents.

AXML document model. An AXML document is modeled as a labelled tree with two types of nodes: (*i*) *data nodes*, or regular XML nodes, which can be labelled with either element names or data values (only for leaf nodes); and (*ii*) *service call nodes*. The latters can encode all the information required to access a Web service (URL, operation name, etc.). When a service call is activated, this information is used by the ActiveXML peer to actually invoke the service. Consider the following disjoint infinite sets of labels: $\mathcal{D}$ of data values, $\mathcal{E}$ of element names, and $\mathcal{S}$ of service names. More formally, an AXML document d is denoted by the expression $\langle \tau, \lambda, < \rangle$, where τ is an ordered tree with a finite set N of nodes and a distinguished node *root*, $\lambda : N \rightarrow \mathcal{E} \cup \mathcal{D} \cup \mathcal{S}$ is a labelling function for nodes in N, such that only leaf nodes are mapped to $\mathcal{D}$, and $<$ associates with each node in N a total order on its children. An important subset of N is that of all the service call nodes of d, called SC_d; for every node v in N, if $\lambda(v) \rightarrow \mathcal{S}$, then $v \in SC_d$.

The children of a service call node stand as its *input parameters*; in the example of Figure 2(a), they are represented by "param" elements (we omit this element if the service call has only one parameter, such as in sc2). When a service call node is invoked, its respective subtrees are passed to the Web service, and the invocation result replaces the call node in the document. Strictly speaking, service call nodes are not eliminated from the AXML document, but kept for possible further invocations [4]. Nonetheless, our analysis focuses on a set of service calls to be evaluated *at a given moment in time* in order to materialize an AXML document. Therefore, without loss of generality, we can ignore the fact that service call nodes may remain in the document after their invocation, as long as they do not need to be invoked again during the materialization process. Figure 2 shows the *SwapWorkspace* document: (a) before and (b) after the invocation of the service call sc5. Furthermore, elements resulting from new service invocations have a special *timestamp* attribute to indicate the current snapshot of the document contents. Thereby, users may choose to consider either all the previous results in the document or only the last invocation results for feeding the new service requests (namely, which data elements are going to be used as service inputs).

Notice that both service input parameters and results may be AXML data (i.e. contain service call nodes). Moreover, the input parameters of a service call may be either explicitly provided by nested AXML elements or specified by XPath expressions [65], such as the "company" parameter of sc3 in Figure 2(a). An input parameter defined by an XPath expression represents a query on the elements of the AXML document; such a query is evaluated whenever the service call is invoked and its result is passed in the service request as subtrees of the parameter element.

Materialization strategies in ActiveXML. The materialization of some AXML data can be either explicitly requested by the user or implicitly triggered by queries that requires the (materialized) content of a document. The materialization process always starts at the peer that hosts the corresponding document, which we call the *master peer* (*e.g.*, peer P_1 for the *SwapWorkspace* document in Figure 1). Currently, ActiveXML peers consider neither flexible service execution (with abstract service references) nor system performance to reason about possible evaluation strategies for AXML materialization. Instead, peers can adopt only a *type-driven strategy* [39], which is defined by the user and derived from the analysis of the input and output types of the requested Web services.

For example, to materialize the service call sc9 in Figure 2(a), the user may specify an expected schema for its result, stating that P_1 accepts only regular XML data for an answer to sc9. Then, the user has to ask P_1 to invoke sc9 (and its entire subtree) at either P_4 or P_5, for instance. That is, P_4 (or P_5) would have to invoke both sc9 and sc10, thus sending back to P_1 only the result of sc9. Observe, however, that the user has to determine exactly which peers are going to participate in the materialization process, and how they should exchange AXML data. In fact, the work of [39] makes room for a performance-based approach to guide the materialization process.

For query processing with AXML documents, selection predicates can be used to avoid the execution of irrelevant service calls, as proposed in [3]. In this case, the goal is to materialize only the intensional elements that contribute to the query result. Furthermore, when declarative services are used, some selection predicates can be *pushed* to be processed at the service providers.

4 Service Invocation Constraints in AXML

The relationships between the service calls of an AXML document encode some *explicit and implicit constraints on the invocation of its service call nodes*. These constraints are mostly derived from producer-consumer relationships between service calls, and they cannot be properly represented in the AXML document tree since they rather form a complex graph. In this Section, we present a canonical formalism to represent the invocation constraints of an AXML document into *dependency graphs*, which are the basis of our optimization strategy.

An important type of constraint is expressed by *invocation dependencies*, which occur when a service call takes other calls as input parameters. These are *data dependencies*; there may also exist *service dependencies* [44], namely when some information about the Web service pointed by a call is determined by the result of another call. We focus here on data dependencies, which are defined in Section 4.1. AXML documents may also contain *invocation consequences*, introduced in Section 4.2. We use these different types of constraints to define the *dependency graph* of an AXML document in Section 4.3, where we also analyze issues related to validity and redundancy of these constraints. Going further, we define in Section 4.4 efficient mechanisms to update the dependency graph with intensional answers at runtime.

4.1 Precedence Constraints

Invocation dependencies represent precedence constraints on the materialization of some AXML data. Namely, one can determine that some service calls must be invoked *before* a given call, because the latter consumes their results. There are two types of such parameters: *concrete* and *non-concrete*, which are defined in the following.

Concrete parameters. A first class of AXML invocation dependencies is attained by *nesting a service call in a parameter of another service call* – namely, by specifying a concrete parameter. For example, in Figure 2(a), service calls sc2 and sc3 are nested as input parameters of sc1. Such a node nesting entails that the call to CheckSwapStatus *depends on* both the call to GetCurrentSwaps and GetSwapLimit. Next, we define this relationship.

DEFINITION 1 *Let d be an AXML document, and v_i be a node in SC_d. A node v_j is a* concrete parameter *of v_i iff:*

- *$v_j \in SC_d$ and v_j is a descendant node of v_i in d; and*
- *there is no node $v_x \in SC_d$ such that v_x is both a descendant of v_i and an ancestor of v_j in d.*

We refer to the set of all the concrete parameters of v_i as $\nabla(v_i)$, where $\nabla(v_i) \subseteq SC_d$.

Furthermore, we use the notation $|A|$ for the cardinality of set A; thus, the number of nodes of SC_d is denoted by $|SC_d|$. Checking whether a service call node is a concrete parameter of another call is rather trivial, and it can be done in advance by a static analysis, when the document is loaded and/or updated by an ActiveXML peer.

Non-concrete paramenters. Users may also specify input parameters that are provided by XPath queries, such as the "foreign_currency" parameter of sc7 in Figure 2(a). In this case, sc4 is not explicitly nested in sc7; it has actually no ancestor relationship with sc7. Yet, materializing sc7 depends on the invocation of sc4, since sc4 contributes to the input of sc7. This yields another class of invocation dependencies, as follows.

DEFINITION 2 *Let d be an AXML document, v_i a node in SC_d, and Q an XPath expression within an input parameter of v_i. A node v_j is a* non-concrete parameter *of v_i iff:*

- *v_j is in the result of $\Phi(Q)$, where $\Phi(Q)$ is a function that returns the set of all the service calls that contribute to Q in d, such that $\Phi(Q) \subseteq SC_d$; and*
- *$v_j \neq v_i$ and there is no node v_x in $\Phi(Q)$ such that v_x is an ancestor of v_j in d.*

The term $\vec{\nabla}(v_i)$ denotes the set of all the non-concrete parameters of v_i, where $\vec{\nabla}(v_i) \subseteq \Phi(Q)$.

Conversely to concrete parameters, a sophisticated analysis may be necessary to compute the set of service calls that contribute to a given XPath expression, such as in [3]. Notice that the XPath expression has to be evaluated first in order to obtain concrete inputs, and then invoke the service. Namely, non-concrete parameters can only be determined *after their respective query is evaluated.* Also, they may imply dependency cycles, possibly leading to an invocation deadlock. To detect this problem, we define in Section 4.3 some validity criteria for the dependency graph of an AXML document. When these cycles are detected, we assume they are either broken prior to our optimization analysis, as in [3], or the materialization process is aborted.

Transitive dependencies. In Definition 2, we disregard some redundant dependencies that may occur in the same

attribute. Basically, such a redundancy happens when some service calls in $\Phi(Q)$ are nested in the AXML document. For example, suppose the "text" and "input_format" parameters of sc9 in Figure 2(a) are rewritten as:

<param name="text"><xpath> // </xpath></param>
<param name="input_format">XML</param>

Then, potentially all the other service calls in the document can contribute to the result of the XPath expression in "text" (assuming that self-dependencies and cycles are properly eliminated). In particular, both sc1 and sc2 are in $\Phi(//)$. However, according to Definition 2, only sc1 is a non-concrete parameter of sc9, since sc2 is nested in sc1. The rationale behind this simplification is that the redundant parameter no longer exists (in its active form) after its first execution, which is triggered by the dependencies of its closest outer service call. In Section 4.3, we use this principle to reduce redundancy of an entire AXML document.

Redundant dependencies are essentially related to the transitivity of service call parameters, which we define next.

DEFINITION 3 *Given two service call nodes v_i and v_j, we say that v_j is an* intensional parameter *of v_i, denoted by $v_j \rightarrow v_i$, iff $v_j \in \nabla(v_i) \cup \vec{\nabla}(v_i)$. The term $fanIn(v_i)$ represents the number of intensional parameters of v_i, which corresponds to $|\nabla(v_i) \cup \vec{\nabla}(v_i)|$. Similarly, the term $fanOut(v_j)$ denotes the number of nodes which have v_j as an intensional parameter.*

DEFINITION 4 *Given two service call nodes v_i and v_j of an AXML document d, the node v_j is a* transitive parameter *of v_i, denoted by $v_j \overset{*}{\rightarrow} v_i$, iff there is a path of the form $v_j \rightarrow v_{x1} \rightarrow \ldots \rightarrow v_{xn} \rightarrow v_i$ in d, with $n \geq 1$.*

The transitive parameter with the largest path length determines the *abstract critical path* of an AXML document, that is the longest path of sequential service invocations. Notice we are considering only the number of service call nodes in the path. When materializing a document, another important path is that with the longest execution time.

4.2 Consequence Constraints

Another way to express invocation constraints in AXML documents is specifying a "followed-by" attribute in the service call elements. For example, in Figure 2(a), this clause constrains the invocation of the service call sc1 with respect to sc5. Namely, once sc5 is invoked, an invocation of sc1 must be triggered *immediately after* (as a consequence of) the invocation of sc5. However, invoking sc1 should not disturb sc5.

Notice the semantics of consequence constraints differs significantly from that of invocation dependencies. While these dependencies are intrinsicaly associated to data flows to feed Web service inputs, consequence constraints denote an invocation sequencing of service calls that do not necessarily have data dependencies between each other.

DEFINITION 5 *An AXML document with collateral calls is of the form $\langle d, \hookrightarrow \rangle$, where d is a regular AXML document and $\hookrightarrow$ is an one-to-one partial mapping of nodes in SC_d to nodes in SC_d. Given two service call nodes v_i and v_j, $v_i \neq v_j$, if $v_i \hookrightarrow v_j$, then each invocation of v_i automatically triggers an invocation of v_j, which is referred to as a* collateral call *of v_i.*

For simplicity, we assume that a service call may be associated with only one collateral call. Observe that collateral calls may trigger other calls, which in turn may have their own collateral calls, and so on. Hence, the invocation consequences of an AXML document may determine transitive constraints, as defined next.

DEFINITION 6 *Let v_i and v_j be two service call nodes of an AXML document $\langle d, \hookrightarrow \rangle$. The node v_j is a* transitive collateral call *of v_i, denoted by $v_i \overset{*}{\hookrightarrow} v_j$, iff there is a path of the form $v_i \hookrightarrow v_{x1} \hookrightarrow \ldots \hookrightarrow v_{xn} \hookrightarrow v_j$ in d, with $n \geq 1$.*

It is worth mentioning that one cannot express collateral calls by using only invocation dependencies. Invoking an outer call always implies the previous materialization of its dependencies. On the contrary, executing nodes that are collateral calls should not disturb their respective counterparts. In general, the distinction between process dependencies and consequences can be also found in workflow specifications, such as the **precede** and **enable** constructs of the *ActivityFlow* language [34]. This is also similar to the *enabling flows* of the Vortex system [30].

The invocation constraints of an AXML document correspond to some basic workflow patterns [54]. More precisely, invocation dependencies can be mapped to the *synchronization* control pattern. In particular, shared dependencies can also produce *parallel splits*. On the other hand, collateral calls correspond to the *sequence* control pattern. Additionally, they calls allows multiple instances of services calls without synchronization, which corresponds to the structural pattern 12 of the classification in [54].

Observe that, when optimizing the materialization of an AXML document, intensional parameters represent essentially some clustering criteria to reduce communication costs. Conversely, collateral calls correspond to the sequential invocation of new service calls possibly without implicit data flows.

4.3 Dependency Graph

We present here a formalism based on *directed acyclic graphs* (DAG) to express the invocation constraints of an

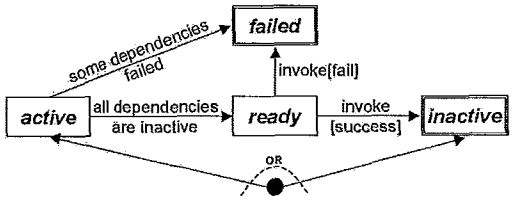


Figure 5. Statechart of service call nodes.

AXML document. We use this abstract representation rather than the AXML tree mainly because non-concrete parameters and collateral calls result into dependencies that can be arbitrarily complex. Such relationships cannot be naturally expressed by tree structures. Furthermore, reasoning about invocation constraints, such as verifying shared dependencies and non-concrete (or collateral) transitivity, is done at this level.

In our analysis of invocation dependencies, we consider that most of the Web services do not understand AXML data, and thus cannot process correctly intensional (concrete and non-concrete) parameters. For regular Web services, these parameters must be invoked before executing the service call. On the other hand, such a invocation might be unnecessary, or even incorrect, for AXML-enabled services. Thus, the user has to specify whether intensional parameters must be invoked *a priori*. In practice, we do not distinguish AXML-enabled Web services, and the execution of intensional parameters is enabled by setting an attribute of their respective "sc" elements. Hence, our analysis is focused on intensional parameters that are always invoked *a priori*.

Figure 5 shows the possible states of a service call node during AXML materialization; the states "*inactive*" and "*failed*" are shown with double rectangles because they are terminal (*i.e.*, when nodes become stable). Service call nodes start with either state "*active*" or "*inactive*". A node may be initially set as "*inactive*" due to many reasons, such as bad AXML specification, user selection, or because some preliminary analysis has considered the invocation of this node unnecessary for the document materialization, as in [3]. On the other hand, an active node becomes "*ready*" when all of its intensional parameters are "*inactive*". If ready, a node can be invoked, and then it reaches either the "*inactive*" state or the "*failed*" state, according to the success or fail, respectively, of its invocation. Also, a node moves to the "*failed*" state if the invocation of some of its dependencies fails.

First-level service calls. Some service call nodes play a distinguished role in the AXML materialization process because the results of their invocation constitute the contents of the AXML document. These results must be kept in the document after its materialization finishes. This is opposed to the results obtained from nested calls, which are often temporary and only needed to invoke their dependant calls. Service call nodes with persistent results are defined next.

DEFINITION 7 *A node v_i is a* first-level service call *of an AXML document d iff $v_i \in SC_d$ and for any node v_x in d, if v_x is an ancestor of v_i, then $v_x \notin SC_d$. We denote by ξ the set of all the first-level service calls of d, such that $\xi \subseteq SC_d$.*

First-level calls can be found statically by a straightforward procedure. In Figure 2(a), the nodes sc1, sc4, sc5, sc8, and sc9 are first-level service calls. Such a distinction is relevant for the optimizer to assess the costs of delegating parts of a document to be materialized by other peers.

Dependency graphs defined. We are now able to formally define the *dependency graph* of an AXML document. This graph concisely represents all the synchronization constraints that must be enforced on the service calls within the document. It is a central input to our optimization effort. Although this graph explicitly refers to invocation dependencies, it also encompasses collateral calls. Our emphasis is on intensional parameters because they reflect essential aspects of the AXML document, while collateral calls come from optional annotations on the service call nodes.

Basically, a dependency graph can be obtained by static analysis. Since users may specify AXML documents with cyclic dependencies and infinite execution loops, first we introduce a general definition for the dependency graph. After that, we restrict the valid dependency graphs to acyclic instances. For that purpose, we have to consider a particular definition of cycles in a dependency graph, since invocation dependencies and collateral calls represent distinguished types of edges between nodes, and they can contribute in a tricky way to form cycles.

DEFINITION 8 *The* dependency graph Δ *of an AXML document $\langle d, \hookrightarrow \rangle$ is denoted by the expression $\left\langle \mathcal{G}, \otimes, \overset{\hookrightarrow}{E}, \epsilon \right\rangle$, where $\mathcal{G}$ is a directed graph with a set V of nodes, $V = SC_d$, and a set E of edges. The set V has a distinguished subset $\otimes$ of* persistent nodes, *such that a node v_i is in $\otimes$ iff either $v_i \in \xi$ or $fanOut(v_i) > 1$. Edges in E are either* simple *or* collateral. *For any two nodes v_i and v_j in V, there is a simple edge $v_j \rightarrow v_i$ in E iff v_j is a intensional parameter of v_i in d. The subset $\overset{\hookrightarrow}{E}$ denotes the collateral edges of E; there is an edge $v_i \hookrightarrow v_j$ in $\overset{\hookrightarrow}{E}$ iff $v_i \hookrightarrow v_j$ in d. The term ϵ denotes a state function that maps each node in V into $\{active, ready, inactive, failed\}$.*

Notice that a dependency graph encodes a *partial order* on the service calls of the respective AXML document. Figure 6 depicts the graph derived from the *SwapWorkspace* document in Figure 2(a). Nodes are represented by circles labelled with service call IDs, where double-line circles are

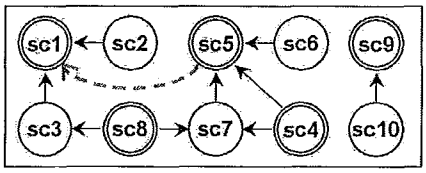


Figure 6. Dependency graph of the *Swap-Workspace* document.

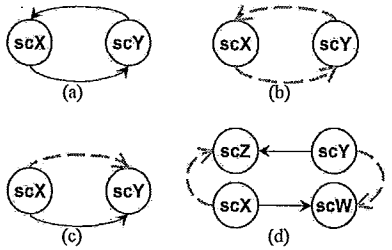


Figure 7. Basic invalid invocation constraints of a dependency graph.

persistent nodes. Dashed arrows indicate collateral edges. We assume all nodes are in the "*active*" state.

Graph validity. Users may specify AXML documents with arbitrary dependency graphs, possibly containing *invocation deadlocks* and *infinite execution loops*. We consider that such documents are invalid, since they cannot be properly materialized.

Figure 7 shows the basic invalid combinations of invocation constraints between service call nodes (illustrated with "scX", "scY", "scW", and "scZ"). Loosely speaking, the subgraph of simple edges of a valid graph must be acyclic, to avoid dependency cycles such as in Figure 7(a). Also, the subgraph of collateral edges should be acyclic, thus preventing the pattern of Figure 7(b). Moreover, an infinite execution loop occurs if either a node is a collateral call of some of its dependencies (Figure 7(c)) or an alternate sequence of simple and collateral edges converge to a cycle, as shown in Figure 7(d). Notice the forbiden patterns of Figure 7 also apply for transitive constraints; for example, if $v_x \xrightarrow{*} v_y$, then we cannot have $v_y \xrightarrow{*} v_x$ in the graph.

In summary, about the interactions between simple and collateral edges, we have that:

- they do not concurr to form deadlock cycles, since collateral calls do not involve data dependencies; but
- their combination may contribute to form infinite execution loops, as shown in Figures 7(c) and 7(d). Hence, to check a dependency graph for execution termination, we have to consider collateral edges as simple edges with opposite direction.

From the above, let the *sequencing component* of a dependency graph be the graph resulting from replacing all the collateral edges by inverted simple edges. Then, we can summarize these validity criteria in the following.

DEFINITION 9 *A dependency graph Δ is* valid *iff:*

- *the subgraph with all the simple edges of Δ is an acyclic digraph; and*
- *the sequencing component of Δ is an acyclic digraph.*

The first bullet of Definition 9 avoids invocation deadlocks, while the second addresses infinite executions. We clearly distinguish between deadlock and termination detection because we consider the user may need a relaxed notion of graph validity, where termination is guaranteed by some pre-determined fixpoint.

Graph validity can be checked when documents are created or altered by the user. However, such a verification may also be required during AXML materialization, at runtime, in case of intensional answers. Checking for validity can be done in $O(|V|^3)$, based on the time complexity of computing the transitive closure of the graph using the well-known Warshall's algorithm [11]. For AXML optimization, we consider only valid graphs.

Redundant dependencies. An AXML document may derive a dependency graph with redundant dependencies. Notice that we can suppress the edge sc4→sc5 in Figure 6, since sc4$\xrightarrow{*}$sc5 (from sc4→sc7 and sc7→sc5) makes it redundant. The resulting reduced graph is shown in Figure 8. Previously, we eliminated redundancy of non-concrete parameters within the same input parameter of a service call. We extend this idea to the entire document.

Following [50], we say that two graphs are equivalent if they are *bisimilar*. However, [50]considers that only edges are labeled. Here, we also consider node labels, and we focus bisimulation on redundant invocation dependencies, as stated next.

DEFINITION 10 *A dependency graph Δ_a* subsumes *another dependency graph Δ_b, denoted by $\Delta_a \subset \Delta_b$, iff:*

- *$V_a = V_b$, $\otimes_a = \otimes_b$, $\overleftrightarrow{E_a} = \overleftrightarrow{E_b}$, $\epsilon_a = \epsilon_b$, and $E_a \subset E_b$; and*
- *there are some nodes v_i and v_j in V_b, such that if both $v_i \rightarrow v_j$ and $v_i \xrightarrow{*} v_j$ are in E_b, then $v_i \rightarrow v_j$ is not in E_a. The edge $v_i \rightarrow v_j$ is said a* redundant dependency *of Δ_b.*

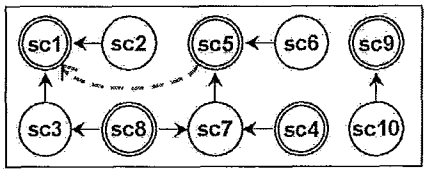


Figure 8. Reduced version of the dependency graph of Figure 6.

The minimum reduced graph *of* Δ_a *is a graph* Δ_a^{MR} *such that* $\Delta_a^{MR} \subset \Delta_a$ *and* Δ_a^{MR} *has no redundant dependency. Moreover, we say that the graphs* Δ_a *and* Δ_b *are* bisimilar *if* $\Delta_a \subset \Delta_b$ *or* $\Delta_b \subset \Delta_a$.

Several redundant dependencies may occur in an AXML document, and their reduction is important to avoid unnecessary optimization efforts. Notice bisimulation is a formal basis for the elimination of redundant AXML dependencies. Therefore, the minimum reduced graph is a *canonical representation* of the invocation constraints of an AXML document, as follows.

PROPOSITION 1 *Let* Δ *be a dependency graph with redundant dependencies. There is at least one reduced graph* Δ^R *which is equivalent to* Δ *and at most one minimum reduced graph* Δ^{MR}.

Proof: By definition of graph bisimulation, if Δ has redundant dependencies of the form $v_i \rightarrow v_j$ and $v_i \stackrel{*}{\rightarrow} v_j$, then we can eliminate at least one $v_j \rightarrow v_i$, and the reduced graph Δ^R remains equivalent to Δ. Furthermore, one can always reduce Δ by applying a sequence of one-edge eliminations, such that if Δ^R cannot be further reduced, then $\Delta^R = \Delta^{MR}$. The ultimate set of eliminated edges is the same regardless of the order of the eliminations steps, and therefore there is only one possible Δ^{MR}.

A dependency graph with only concrete parameters is reduced by definition, since redundancy is intrinsically related to shared dependencies (which occur due to non-concrete parameters). On the other hand, for non-concrete parameters, graph reduction has time complexity $O(|V|^3)$, based on the transitive closure of the graph. However, only nodes with $fanOut \geq 1$ can be origin of redundant edges, and the tight bound is actually $O(|\otimes| \times |V|^2)$. Notice that non-concrete parameters are, in general, quite expensive to be handled in AXML materialization.

Unless stated otherwise, hereafter we will use the terms dependency graph and minimum reduced graph interchangeably.

Exit points. Nodes that do not have outgoing simple edges (*i.e.*, with $fanOut = 0$) are particularly important for the materialization of a dependency graph; they are said the *exit points* of the graph. They can be used to unfold a graph into spanning trees, thus enabling the optimizer to break the materialization problem into smaller parts and thereby to reduce the overall complexity. Also, they represent points where the materialization process finishes (*i.e.*, after materializing them and properly triggering their collateral calls, the materialization process should stop). For example, the nodes sc1, sc5, and sc9 are the exit points of the graph in Figure 6.

LEMMA 1 *Every valid non-null dependency graph* Δ *has at least one exit point.*

Proof: The crux here is that, although collateral edges may concur with invocation dependencies to cause cycles, they have no influence on exit points (to determine the $fanOut$ of a node). Thus, only simple edges must be considered. First, suppose Δ is a singleton, namely $V = \{v\}$. Then, $fanOut(v) = 0$ by definition, and v is an exit point. Consider now that Δ has $|V|$ nodes, with $|V| > 1$, such that each node has at least one simple outgoing edge. Pick any node v_1 in Δ; since $fanOut(v_1) > 0$, there is a node v_2 such that $v_1 \rightarrow v_2$. In turn, v_2 also has an outgoing edge $v_2 \rightarrow v_3$, and so forth. The nodes in Δ are finite, and this path cannot continue forever. At some point, this path leads to repeated nodes, thus constituting a cycle. Since Δ is valid, hence an acyclic digraph w.r.t. simple edges, this is a contradiction. Therefore, there must exist at least one node without outgoing simple edges for Δ to be valid.

Lemma 1 is important because it guarantees that, despite the complexity of the shared dependencies and collateral calls, the optimizer has always an exit point to start evaluating a valid dependency graph. Furthermore, assuming service executions always stop after some period of time, valid graphs are termination-safe, as shown in the following.

PROPOSITION 2 *In a valid dependency graph* Δ*, given any node* v*, either* v *is an exit point of* Δ *or there is a finite path between* v *and some exit point* v_e *of* Δ*, such that* $v \stackrel{*}{\rightarrow} v_e$*. Moreover, all the transitive collateral calls that originate directly or indirectly from nodes in* $v \stackrel{*}{\rightarrow} v_e$ *(including* v *and* v_e*) have a finite path.*

Proof: If Δ is a singleton, then v is an exit point by definition. On the other hand, consider Δ has $|V|$ nodes, with $|V| > 1$, such that each node has an arbitrary number of both simple and collateral outgoing edges. First, from Lemma 1, we have that Δ has at least one exit point. Furthermore, picking any node v in Δ, if $fanOut(v) = 0$, then v is an exit point. Otherwise, there is at least one node v_x such that $v \rightarrow v_x$. In turn, either v_x is an exit point or it also has an outgoing edge $v_x \rightarrow v_y$, and so forth. Since Δ is cycle-free w.r.t. simple edges, by induction this path has

to lead to some exit point v_e. Moreover, $|V|$ is finite, hence the lenght of the transitive dependency $v \xrightarrow{*} v_e$ is also finite. Consider now the sequencing component of Δ. Any collateral call that is triggered (directly or not) by some node in $v \xrightarrow{*} v_e$, including v and v_e, must be in a sequencing path that leads to v_e (recall collateral edges are inverted in the sequencing component). Notice that such a path does not end with v_e if this node has a collateral call. However, the sequencing component of Δ is an acyclic digraph with a finite number of nodes, and therefore all of its sequencing paths are finite.

Notice Proposition 2 concerns the *snapshot semantics* of a dependency graph. That is, it does not consider the effects of occasional intensional answers, but rather the current graph topology. Next, we discuss how graphs can evolve.

4.4 Dynamic Graph Updates

An AXML-enabled system may allow Web services to return service calls in their results. This artifice can be very useful in many scenarios. For instance, suppose a Web service does not have a certain information that was requested by the user, but it knows which Web services can provide it. In this case, the service may return other calls (to alternative providers), and let the user decide whether to pursue the request through other Web services. In this way, the materialization of AXML data can be *dynamically distributed*, thus providing peers with great flexibility for collaboration [32].

In the *CurrencySwap* example (Figures 1 and 2), suppose the repository of PDF files at P_4 is down, and invoking the service call sc10 at P_4 returns the following result:

```
<sc id="11" service="Ps2Pdf">
  <sc id="12" service="GetContractPS"/></sc>
```

Then, to materialize the *SwapWorkspace* document, the master peer has now to invoke both sc11 and sc12, to gather input data for the "text" parameter of sc9. Moreover, this may require discovering information about the peers that provide the Web services referred by sc11 and sc12.

Intensional aswers may significantly change the AXML document – indeed they raise several tough problems for AXML optimization:

- as these answers arrive, the dependency graph has to be updated (at runtime) with the new service calls to allow checking for validity;
- it may be necessary to refresh the service directory with information about the requested Web services. Also, optimizing the newcomers may involve gathering some statistics and costs parameters;
- since the specification of the AXML document is altered, involving new data flows and possibly other service providers, previous optimization choices may be contradicted; and
- some Web services might return undesirable or infinite intensional answers.

To guarantee a correct AXML materialization (in terms of data types) and to ensure its termination, the ActiveXML system implements a powerful typing mechanism for service call results [39]. Intensional results are recursivelly materialized until either the document satisfies a specific type or a fixpoint is reached. In this paper, we rather analyze intensional answers from a performance-oriented perspective. Hence, we consider issues related to dynamically updating dependency graphs with these answers, and their impact on the optimization process.

Connecting intensional answers. The idea behind intensional answers is that the AXML document may "evolve" during the materialization process. Consequently, each intensional answer requires an update operation on the dependency graph. Recall that intensional nodes remain either inactive or failed after invoked. A dependency graph update is defined as follows.

DEFINITION 11 *Let d be an AXML document, Δ its dependency graph and v_i a node of Δ. Suppose d_u is the AXML data returned by the invocation of v_i, which is used to update d. If the graph Δ_u obtained from d_u is not null, then Δ_u is said an* intensional answer *of v_i.*

DEFINITION 12 *An* update operation *u is a triple $\langle \Delta, v_i, \Delta_u \rangle$, where Δ is a dependency graph containing the node v_i, and Δ_u is an intensional answer of v_i to be inserted into Δ. The operation u transforms Δ into a graph $\Delta' = \left\langle \mathcal{G}', \otimes', \overset{\hookrightarrow}{E}', \epsilon' \right\rangle$ according to the Update function in Figure 9(a). The node v_i is said the* origin *of u.*

To update the dependency graph, intensional answers must be properly connected to it, such that invocation constraints and their transitive relationships are preserved. In general, propagating these constraints may be very costly. Fortunately, in practice, we find some heuristics that are particularly handy in this context. Based on some properties of the AXML document model, we make the following assumptions:

(*i*) *New service call nodes are not affected by the collateral relationships of the AXML document.* The intuition is that collateral calls point to specific service calls references. Also, they represent new instances of service invocations, regardless of previous execution results. Therefore, the newcomers neither inherit the collateral relationships of their origin node nor are referred by pre-existing nodes;

```
1  function Update(Δ, v_i, Δ_u): Δ'
2  {Update Δ at node v_i with Δ_u.}
3  begin
4    let V' = V ∪ V_u        ↪  ↪  ↪
5    let E' = E ∪ E_u and E'=E ∪ E_u
6    if v_i ∈ ξ then {first-level call}
7      ξ' = ξ ∪ ξ_u
8    else ξ' = ξ
9    end if
10   if fanOut(v_i) > 1 then {shared dependency}
11     ⊗' = ⊗ ∪ ⊗_u ∪ ξ_u
12   else ⊗' = ⊗ ∪ ⊗_u
13   end if
14   ConnectSubGraph(Δ_u, v_i, Δ')
15   Re-evaluate non-concrete parameters of Δ'
16   for each v_x in V' do
17     set ε'(v_x) according to ε(v_x) or ε_u(v_x)
18   end for
19   set ε'(v_i) = inactive
20   return Δ'
21 end

21 procedure ConnectSubgraph(Δ_u, v_i, Δ')
22 {Connect the subgraph Δ_u to Δ' according to
    v_i's dependencies.}
23 begin
24   let Out be the set of outgoing "→" edges of v_i,
       such that |Out| = fanOut(v_i)
25   for each v_x in ξ_u do
26     for each v_i → v_y in Out do
27       Add v_x → v_y to E'
28     end for
29   end for
30 end
```

(a)

```
1  procedure ConnectSubgraphWithPipe(Δ_u, v_i, Δ')
2  {Connect the subgraph Δ_u to Δ' according to
    v_i's dependencies, using a pipe node.}
3  begin
4    Add a new node pipe to V'
5    for each v_x in ξ_u do
6      Add v_x → pipe to E'
7    end for
8    for each v_i → v_y in E do
9      Add pipe → v_y to E'
10   end for
11   if v_i ∈ ξ then
12     Add pipe to ξ'
13   end if
14   if fanOut(v_i) > 1 then
15     Add pipe to ⊗'
16   end if
17 end
```

(b)

Figure 9. Algorithm to (*a*) update Δ and (*b*) connect Δ_u through a *pipe* node.

(*ii*) *Intensional answers do not contain non-concrete parameters refering to nodes from other parts of the AXML document*, because Web services are not necessarily aware of the document contents, and these parameters involve context-dependant specification;

(*iii*) Analogously, *the collateral calls of intensional answers refer only to newcomers*; and

(*iv*) *New nodes do not depend on the intensional parameters of the origin call*, since invocation results either replace the entire subtree of their origin node or are placed as their siblings in the document tree.

From these points, we can consider that intensional answers are *loosely coupled* with the dependency graph. Therefore, only the outgoing simple edges of the origin service call are used to connect intensional answers, as encoded in the algorithms of Figure 9.

We show an example of connecting intensional answers to a dependency graph in Figure 10. In this example, a service call node "scX" is invoked and returns some arbitrary intensional answer represented by Δ_u (Figure 10(a)). Then, for each outgoing edge of "scX", we connect each first-level service call of Δ_u to the respective target node. Namely, to the node depending on "scX", as illustred in Figure 10(b).

Pipelined graphs. Despite these simplifications, updating a dependency graph may involve creating many edges to link the new nodes. The first-level calls of the intensional answer must be joined to the service calls that depend on the origin node (see the double loop in lines 25 and 26 of Figure 9(a)). To improve the performance of this procedure, we introduce a special service call node in the resulting graph. This node refers to a very simple Web service called "*pipe*", which is a generic service – that is, it can be executed by (potentially) any peer of the system.

The semantics of the *pipe* service is fairly simple: it receives the results from its invocation dependencies, and passes them to its dependant nodes. *Pipe* nodes are transparent with respect to the contents of their inputs; they neither produce new AXML data or eliminate any node. Their goal is to simplify the changes that are necessary to update a dependency graph.

DEFINITION 13 *Given a dependency graph Δ and an update operation u, the* pipelined graph Δ^p *is the graph resulting from u such that a pipe node is used to connect Δ_u into Δ, according to the algorithm of Figure 9(b).*

Notice in Figure 10(c) that a *pipe* node concentrates the edges that connect an intensional answer, thus reducing the memory requirements of the resulting graph. If new nodes are connected directly to the dependency graph, then this number is given by:

$$fanOut(v_i) \times |\xi_u| \quad . \tag{1}$$

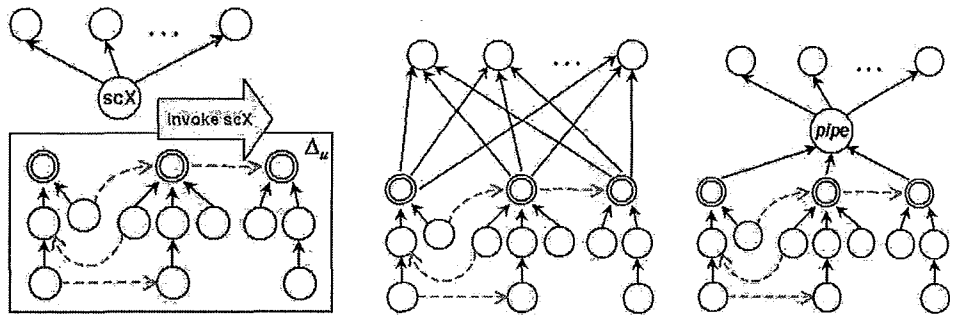


Figure 10. (a) The invocation of node "scX" returns an intensional answer Δ_u; (b) connecting Δ_u to the dependency graph with the *Update* algorithm; and (c) using a *pipe* node.

On the other hand, with *pipe* nodes, the number of edges to link the intensional answer Δ_u of a node v_i is bound to:

$$fanOut(v_i) + |\xi_u| \quad . \tag{2}$$

Pipelined graphs are particularly useful when the fan-out of the origin call is greater than 1. Also, they are helpful if it is necessary to relax the loosely-coupling assumption. That is, they can be used to "centralize" properties inherited by the new nodes from their origin. For instance, if one considers that intensional answers should inherit the collateral calls of their origin node. Furthermore, it can be proved that the graph Δ^p is equivalent (according to Definition 10) to Δ' augmented with the *pipe* node and its edges. Pipe nodes are inspired on the use of "*epsilon edges*", which are introduced in [50].

Update validity. Updating a dependency graph requires checking whether the intensional answers lead to deadlocks or infinite collateral loops. Therefore, we state the following criterion.

DEFINITION 14 *An update operation $u = \langle \Delta, v_i, \Delta_u \rangle$ is* correct *iff its resulting graph Δ' is valid.*

Observe that our approach allows to verify the validity of the resulting document before applying the changes to it, by reasoning based on dependency graphs. The heuristics used to update the graph also contribute to restrict such a verification to only the intensional answers. This is possible because we consider that new intensional nodes are not tightly connected to the AXML document.

PROPOSITION 3 *Let Δ be a valid dependency graph and $u = \langle \Delta, v_i, \Delta_u \rangle$ be an update operation. If Δ_u is valid, then u is correct.*

Proof Sketch: By definition (according to *the Update* algorithm), an update operation connects new nodes such that: they do not depend on existing service calls; and they do not trigger (or are triggered by) collateral calls of the original graph. Therefore, the newcomers can contribute to form neither invocation deadlocks nor collateral cycles with existing nodes. Since the graph is valid before the update, only its new portion needs to be checked. Hence, the validity of Δ' is determined by Δ_u.

Lenient updates. Another performance issue of handling intensional answers can be found at line 15 of the *Update* algorithm, in Figure 9(a). Whenever the AXML document changes, the input queries of non-concrete parameters may need to be re-evaluated. This approach is often quite time consuming. A less expensive alternative to do that consists in performing a *lenient analysis*: the re-evaluation is triggered only after a certain number of service call invocations (or intensional answers), thus allowing the document to evolve freely meanwhile. To proper formalize this idea, we first define the high-level semantics of instantiating an AXML materialization process as follows.

DEFINITION 15 *Let d be an AXML document, and suppose the dynamic sequence $I_{timeline} = [v_1, \ldots, v_n]$ is obtained by successively picking a ready node of SC_d and invoking it, until all nodes in SC_d are stable, where v_n is the last node invoked. A* materialization phase *ϕ of d during $I_{timeline}$ is an expression $\langle \Delta_0, \wp, IT, U, \beta \rangle$, where:*

- *Δ_0 is the dependency graph of d before v_1 is invoked;*
- *$\wp$ is the* duration criterion *of ϕ, which is a boolean predicate defined on some properties of d and ϕ, such as the maximum number of invoked service calls. The duration criterion is evaluated for each v_x in $I_{timeline}$, $1 \leq x \leq n$, until it becomes true;*

- *IT is the* invocation trail *of ϕ, which consists of a sequence $[v_1, \ldots, v_\ell]$ of service calls invoked until $\wp$ is evaluated to true, such that $IT \subseteq I_{timeline}$. The number ℓ is said the* duration *of ϕ, where $1 \leq \ell \leq n$;*

- *U is a sequence of update operations caused by the service invocations of IT on Δ_0; and*

- *β denotes the* backlog *of ϕ, namely a surjection from nodes in IT to new service call nodes in SC_d, such that $\beta(v_x)$ returns the set of all the active nodes that were added to SC_d by intensional answers from the invocation of v_1 to v_x, with v_x included and $1 \leq x \leq \ell$.*

We denote by Δ_x the snapshot graph *of d after the invocation of v_x, $1 \leq x \leq \ell$. The phase ϕ is* incomplete *while $\wp$ is false and SC_d has active nodes; otherwise, ϕ is* complete.

The semantics of a materialization phase is a period, measured in number of service call invocations, while the contents of an AXML document evolve. Notice $I_{timeline}$ may be infinite. For example, suppose the AXML document contains a service call that always returns another call to the same service. Also, the duration criterion may be defined on physical properties of the materialization phase, such as the total time spent in the execution of the service calls. Furthermore, only active nodes are accounted in the backlog of a phase, since some new intensional nodes may become inactive (by being invoked) before a phase finishes.

DEFINITION 16 *A materialization phase ϕ of length ℓ is* admissible *iff the snapshot graph Δ_ℓ is valid. We say that ϕ is x-*admissible *iff Δ_ℓ is invalid and there is a number x, $1 \leq x < \ell$, which is the maximum value for that Δ_x is valid.*

COROLLARY 1 *Given any phase ϕ with a valid Δ_0, if all the update operations in U are correct, then ϕ is admissible.*

Based on materialization phases, we can determine checkpoints for the re-evaluation of non-concrete parameters, thereby deferring this analysis to a "reasonable" amount of changes. To perform such an analysis, we consider a *lenient update operation* by excluding line 15 from the *Update* algorithm (Figure 9(a)), and we extend Definition 15 as follows.

DEFINITION 17 *A* lenient materialization phase *is denoted by $\phi^{len} = \langle \Delta_0, \wp, IT, U^{len}, \beta, len \rangle$, where:*

- *The terms Δ_0, $\wp$, IT, and β are defined as for ϕ;*

- *U^{len} is a sequence of lenient update operations caused by the service invocations of IT on Δ_0; and*

- *len is the* leniency criterion *of ϕ, which is a boolean combination of predicates of the form "$|A|$ op a", such that a is an integer, the term op is in $\{=, >, <, \leq, \geq\}$, and the set A is in $\{SC_d, IT, U^{len}, \beta\}$. At each v_x in IT, $1 \leq x < \ell$, this criterion is checked, and the re-evaluation of the non-concrete parameters of Δ_x is triggered if len is true.*

Additionally, the re-evaluation is always triggered after v_ℓ.

Conversely to $\wp$, we restrict the definition of *len*, since a lenient re-evaluation analysis focuses rather on amounts of service calls (invoked or not). An interesting property of ϕ^{len} is that, at each re-evaluation point, the analysis has to consider only the nodes in the backlog in order to insert new non-concrete dependencies into the graph.

Furthermore, recall the XPath expression of an input parameter is always evaluated when the respective service call is invoked. If some intensional nodes contribute to this query, then they are considered non-concrete parameters and the results of their invocation are passed to the outer service call. Hence, although the dependency graph may miss some non-concrete edges in a lenient analysis, these dependencies are always enforced during the materialization. Observe that Proposition 1 also applies to lenient phases. The major drawback of a lenient analysis is that peers cannot attempt to optimize in advance *some* of the data transfers related to service calls in the backlog.

Final remarks. Representing service invocation constraints is an important issue in the materialization of AXML documents. In particular, because it enables the system to check relevant properties of a document, and to control its evolution during the materialization process. In spite of that, it is worth mentioning that the canonical model and update techniques that we propose in this Section are not a requirement of the XCraft optimization strategy. They actually provide a formal underpinning that can be explored by any systematic approach to AXML verification and/or optimization. Nonetheless, since our proposal relies on a very popular structure for components dependencies (*i.e.*, graphs), the cornerstone ideas of our optimization strategy can be applied on a general context, regardless of the techniques presented in this Section.

In the next Section, we formalize the problem of determining an efficient configuration of service executors and service callers to materialize an AXML document.

5 Materializing AXML Documents

A basic way to write an AXML document consists in hard-wiring a specific address for each embedded service call, including the service URL. Namely, the user has to locate a peer to execute each service call. This approach is

quite cumbersome, specially because P2P systems are often complex, highly-dynamic arrangements of many peers. In a more flexible approach, one can use abstract references to identify Web services, based in some ontology of services, such as in OWL-S [41]. These references correspond to entries in a service directory, as in a UDDI repository [52], where they are mapped to the peers that actually provide the services. Since a Web service may be provided by several peers, ideally users can rely on the system to choose the best provider (in terms of performance, or another set of properties) to execute each service call.

Notice that even if specific service addresses are used, peers can collaborate to materialize a document by delegating some service calls to be invoked at other peers. In this case, peers need some strategy to distribute the materialization process. Also, since such a collaboration allows many different materialization alternatives, an automatic strategy requires metrics to reason about delegation.

In this Section, before enumerating these alternatives and pointing appropriate metrics, we describe the main participants of the AXML materialization process, how they can collaborate in a P2P scenario, and their necessary bindings for Web service invocation. Then, in Section 5.4, we characterize the problem of optimizing AXML materialization: we determine some bounds for the search space, introduce the notion of materialization plans for AXML documents, and discuss important issues on generating and ranking these plans. We close this discussion in Section 5.5 with an analysis of the impact of the dependency graph shape on optimization strategies for AXML materialization. In the sequel, we assume an infinite set $\mathcal{P}$ of peer names in the system. Furthermore, each peer keeps a list of *neighbors* (denoted by $\mathcal{N}$, such that $\mathcal{N} \subseteq \mathcal{P}$), namely the peers it can collaborate for AXML materialization.

5.1 Equivalent, Replicated and Generic Services

The evolution of the Semantic Web has fostered services orchestration in several distributed scenarios, such as P2P systems [8, 17, 26] and grid computing [14, 21, 29, 57, 66]. An atractive property of semantic-enabled systems is that distributed applications can be defined by *abstract specifications*, and then instantiated on an execution environment based upon *equivalent Web services*. A catalog of services and some matchmaking mechanism are required to enable service selection [19, 23, 36]. This approach has many advantages, and has been widely explored. In P2P systems, *several peers are expected to provide the same Web service.*

To materialize an AXML document with abstract service references, it is important to determine which peers shall be considered to participate in the process, since the Web service requested by each service call node may be provided by several peers. A simple case of service equivalence is when *Web services are arbitrarily replicated* in the system. Replication of data and/or Web services is typically employed in distributed scenarios to increase throughput and reliability [12, 14]. In the ActiveXML framework, service replication is particularly easy for *declarative Web services*, which are defined by XML queries: *any* peer can evaluate a query (and thus become a service provider), as soon as it has the data on which the query applies. A mechanism for declarative service replication in ActiveXML has been developed in [7]. Another class of services likely to be deployed on several peers consists of *generic services*, such as encryption and data compression services. Usually, these services do not change the contents of the data they operate on, but act on some orthogonal aspects, such as its size and its encryption status.

The AXML optimizer may be free to *dynamically deploy* declarative or generic services to some peers during the materialization process, if this allows to improve the overall materialization performance. However, dynamic service replication may significantly increase the number of AXML materialization alternatives. Hence, the optimizer must consider whether enlarging the search space with these possibilities makes the problem too complex.

We extend the basic AXML document model to allow service call nodes to mention the symbol "any" as the servers providing a given Web service. The semantics is that *the user does not impose any specific provider for this invocation*; any server that the optimizer may find is considered good. For simplicity, we assume that any $\in \mathcal{N}$. Also, we consider the symbol "unknown", which indicates that the optimizer does not hold information about a requested service, but can lookup for it in the P2P system. We also assume Web services are organized into classes of equivalent services, and peers can gather information about the distribution of these services on the network (as in [8]). Such an information does not necessarily represent the global status of the system, but only the system visibility from a given peer. A basic capability of an AXML-enabled peer is to identify the providers of a service call node, as follows.

DEFINITION 18 *Given a service call node v, its* execution scope L_v^E *represents the peers that can execute the Web service of v, such that L_v^E = any | unknown | $\{P_1, \ldots, P_n\}$, where any indicates that all the peers in $\mathcal{N}$ can execute v, the symbol unknown denotes that L_v^E is undefined, and $\{P_1, \ldots, P_n\}$ is a finite set of peers in $\mathcal{P}$ which provide the service requested by v, with $n \geq 0$.*

Observe that $L_v^E = \{\}$ means that the optimizer did not discover any information about peers providing the service of v. Since peers can join or leave the system randomly, the execution scope of a service call is rather a snapshot of the system status. Because of that, if L_v^E is empty, the peer can either retry to locate the service afterwards (hoping for some

change in the system) or ask other peers to try to fill in the execution scope of the node. The execution scope may also be determined by the user, through explicit peer addressing. It is worth noting that a service provider does not have to be a neighbor. For example, suppose the neighbors of peer P_1 in Figure 1(a) are $\mathcal{N}_{P_1} = \{P_2, P_4, P_5\}$. Still, according to Figure 1(b), peer P_3 is among the providers of sc1 on the *SwapWorkspace* document at P_1. Next, we discuss ways for peers to collaborate in AXML materialization.

5.2 Exploring P2P Collaboration in AXML

Distributed computing is inherent in AXML materialization, since service executions usually take place in different peers. Going further, many other aspects can also be distributed, such as:

- peers can collaborate to *locate service providers*;
- peers can *delegate parts of the materialization* of a document to other peers; and
- similarly, peers can ask other peers to *generate parts of a materialization plan* for an AXML document.

Basically, AXML materialization can be orchestrated in a P2P system by some message exchanges between peers. In this scenario, the basis of peers interaction is the *materialization plan*, which is the fundamental element used to control the AXML materialization process. Such a plan determines how each service call node is going to be materialized; it can be split, and distributed among peers. Moreover, peers can revise some optimization decisions of a plan, and then choose to re-split it among other peers. Thereby, the materialization process can be spread across the system in a decentralized manner. Materialization plans are formally defined in Section 5.4.

Locating service providers is the simplest type of P2P collaboration. If a peer cannot find information about the execution scope of some service call nodes, then it may decide to send a partially-specified plan to another peer, along with a request to properly annotate such an information on the plan. On the other hand, delegating AXML materialization requires more sophisticated control mechanisms. First, recall that the start point of the materialization process is always at the master peer of the AXML document. Then, the master peer decides whether other peers will be invited to collaborate or not. Delegating the invocation of a service call node v consists in setting another *caller* for v, which is different from the master peer. In particular, the master peer sends a *delegation plan*, which contains v (or more nodes), its input parameters (possibly including its invocation dependencies, if v is not ready), and its collateral call to the caller. In turn, the caller is in charge of triggering the invocation of v (and of its dependencies and collateral calls,

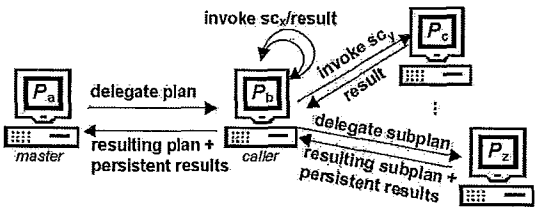


Figure 11. AXML delegation in a P2P system.

if necessary), and gathering its result. The caller must send back the result of v to the master peer, as well as the result of all the persistent nodes that were sent with v. The master peer may also choose to evaluate some of the dependencies of a delegated node before requesting its remote materialization. These interactions are illustrated in Figure 11.

Notice that a delegated part of an AXML document does not necessarily correspond to a subtree, but to some partition of its dependency graph. Reasoning about AXML delegation has to consider the performance impact of changing the caller of each node of the dependency graph. Such a decision is mainly oriented by the data flows between service call nodes. For example, delegation may be beneficial to materialize nodes sc9 and sc10 of the *SwapWorkspace* document, in Figure 3, since it enables to exploit a fast communication link between peers P_4 and P_5. Nevertheless, there may exist constraints on the invocation of a service call node, such as secutity and price policies. Thus, users may also choose to specify exactly which peers are allowed to invoke some service calls.

DEFINITION 19 *Given a service call node v, the* delegation scope L_v^C *denotes the peers that can invoke v, such that $L_v^C = any \mid \{P_1, \ldots, P_n\}$, where any indicates that all the peers in $\mathcal{N}$ can invoke v, and $\{P_1, \ldots, P_n\}$ is a finite set of peers in $\mathcal{P}$, $n \geq 0$.*

In principle, the optimizer may consider any peer in the system to delegate some AXML materialization, accepting that such a peer is allowed to invoke the corresponding service calls. However, this can rapidly make the optimization problem intractable, as we discuss in Section 5.4. Therefore, we assume a *small-world P2P scenario*, where the optimizer may restrict the delegation scope to the set of service providers that are involved in the document materialization. To further limit this, it may also apply the *Context heuristic* [44], such that only selected executors are used to determine the delegation scope. Also, several P2P systems consider the need of some specialized peers, which are more able than others to perform some tasks, such as query routing and data location [40]. Similarly, some peers may be tailored for some AXML materialization tasks, such as locating the execution scope of nodes or executing delegated

plans (*e.g.*, due to their high connectivity with other peers in the system). Therefore, the optimizer may keep a list of such peers, and include them in the delegation scope even if they are not involved in the execution scope, to improve the materialization performance.

A major motivation of AXML delegation is similar to the idea of the *edge-zeroing* algorithms for parallel scheduling [33]. In a dependency graph, these algorithms analyze the edges with high communication cost, and attempt to define clusters of tasks to be run at the same processor. Initially, each node of the graph represents a cluster. Then, at each step of the algorithm, the edge with the largest communication cost is found, and the two clusters incident by this edge are merged if such a merging does not increase the overall performance. However, the constraints on both the execution and delegation scope of each service call node of an AXML document do not allow to arbitrarily assign service executions to peers. Moreover, communication costs are not zeroed if two connected service executions occur in the same peer (though they are really reduced), since SOAP messaging usually involves some additional time-consuming operations, such as parsing and packing the transfered data [44]. Hence, deciding about delegation mostly relies on comparing materialization alternatives.

We assume peers are autonomous, thus the master peer cannot enforce transitive delegation to other peers. Namely, the result of each delegated plan is always sent to the master peer. Considering otherwise would increase significantly an already huge space of AXML materilization alternatives. However, since peers have different perspectives of the system, a peer may decide to reoptimize a delegated plan, possibly by delegating parts of it. Moreover, peers may return a delegated plan intact or partially materialized. In this case, the master peer either reoptimizes the plan, trying to find another peer to delegate the plan, or evaluates it itself. Nevertheless, if a peer decides to reoptimize a plan, then it cannot include the original master peer in the delegation scope of the respective service call nodes. Also, to avoid endless delegations, the original optimizer can determine some limits for plan reoptimizations. These limits and the plan provenance are encoded in the materialization plan.

Distributing the optimization of AXML documents is analogous to delegating materialization tasks, and it can help peers to handle requests overload and insufficient support information. If a materialization problem is too large, the master peer may split the dependency graph without making any considerations about either execution scope or performance, and then send parts of the problem to be solved by other peers. To support P2P collaborations in ActiveXML, we assume each peer has to provide the basic Web services shown in Table 1.

Table 1. Services for P2P collaboration.

Web service	Description
locate	Accepts requests to discover the execution scope for some materialization plan. It returns a (possibly partially-)annotated plan.
optimize	Accepts requests to find an efficient materialization plan for some dependency graph. It returns a (possibly partially-specified) materialization plan.
submit	Receives requests to execute some materialization plan, by possibly re-optimizing it. It returns the plan and its persistent results.

5.3 Enacting AXML Materialization

In the AXML universe, the basic elements of a P2P setting are peers, Web services and AXML documents. Essentially, *peers* are uniquely-identified agents connected through a network, and *services* are operations that peers can perform. A service may require input parameters that are instantiated at runtime. It also has a termination status, such as "*success*" and "*fail*". The *invocation* of a Web service is an event, namely a compact occurence that enables: 1) the flow of input parameters to the peer that is going to execute the service; 2) the service execution; and 3) the transfer of the results to the peer that requested the service. By default, a service call node is invoked by the master peer of its respective document, but this invocation can be delegated to another peer. To be invoked, a service call must have all the necessary information to identify the requested Web service (as defined in the SOAP and WSDL standards [59]). In particular, it must have the address of the peer that is going to execute the service, as stated next.

DEFINITION 20 *Given a service call node v held by peer P_v, an* invocation $\mathcal{I}$ *of v is denoted by the expression $\langle v, P^E, P^C, status \rangle$, where:*

- *P^E and P^C are, respectively, the* executor *and the* caller *of v, such that $P^E \in L_v^E$ and $P^C \in (P_v \cup L_v^C)$;*
- *neither L_v^E is empty nor it contains the symbol "unknown"; and*
- *the status of $\mathcal{I}$ is determined by the termination status of the service requested by v, such that status in $\{success, fail\}$.*

For example, in Figure 3 (center), we have $\langle sc10, P_5, P_4, success \rangle$ and $\langle sc9, P_4, P_4, success \rangle$, assuming both invocations are successifuly executed.

For simplicity, when considering materialization phases, we omitted in Definition 15 the information about callers and executors of each service call invocation. Now, we can

ground an invocation sequence $I_{timeline}$ to a specific location context, as follows.

DEFINITION 21 *Given an AXML document d, a* grounded invocation sequence $I_{timeline} = [\mathcal{I}_1, \ldots, \mathcal{I}_n]$ *is a dynamic sequence obtained by successively picking a ready node v_x in SC_d, setting an $\mathcal{I}_x = \langle v_x, P_x^E, P_x^C, status_x \rangle$ and invoking it, until either $SC_d = \{\}$ or SC_d has no ready node, such that $1 \leq x \leq n$. Moreover, if $v_i \hookrightarrow v_j$ in d, then $\mathcal{I}_j$ is a successor of $\mathcal{I}_i$ in $I_{timeline}$, and the only invocations between $\mathcal{I}_j$ and $\mathcal{I}_i$ are the dependencies of v_i and their collateral calls. A* grounded invocation track *IT of $I_{timeline}$ is of the form $IT = [\mathcal{I}_1, \ldots, \mathcal{I}_\ell]$, with $\ell \leq n$.*

Observe that nodes are invoked in proper order in $I_{timeline}$ (ready nodes first with subsequent collateral calls), and invocation constraints are enforced accordingly. Hereafter, we assume both $I_{timeline}$ and IT to be grounded, unless stated otherwised.

Materializing an AXML document consists in invoking all of its embedded service calls, as well as the occasional intensional answers. This process yields a new version of the document, as defined next.

DEFINITION 22 *Let d be an AXML document and ϕ be a (grounded) materialization phase of d with length ℓ. A* materialized version *of d is another AXML document d' obtained by the service invocations of ϕ. We say that d' is* complete *iff all the nodes in Δ_ℓ are inactive; otherwise, d' is partial. Moreover, the materialization of d into d' is* successful *iff it is complete and $status_i$ = "success" for each invocation $\mathcal{I}_i$ in IT of ϕ, $1 \leq i \leq \ell$.*

If some invocations fail, their dependant calls are not invoked (since they cannot become ready) and the respective failures are reported to the user. Still, we consider that all the remaining service calls are invoked, if possible. Also, we assume that the user is not interested in *undoing* service calls when some of them fail in the materialization of an AXML document. That is, the document does not represent a transaction unit.

The master peer receives document requests and starts their materialization, possibly by delegating parts of the service invocations. After peers finish their materialization tasks, they must send all the persistent service results back to the master peer. In our AXML settings, we assume communication links between peers may have different bandwidth.

5.4 The AXML Optimization Problem

The diversity of Web services providers, P2P collaboration opportunities, peers capabilities, and invocation dependencies allows the materialization of an AXML document to be performed through many different alternatives, with different performance. The *makespan* of a materialization alternative is the time from the materialization starts until the last service call invocation is completed and the required results are returned to the master peer. We adopt this performance metric because it is based on *response time*, which has tipically a significant perceived impact on the user [20, 43].

Optimizing the materialization performance of an AXML document consists in minimizing its makespan. This involves two main issues: (*i*) *planning resource selection*, that is determining a caller and an executor for each service call, such that both service execution and communication costs are minimized; and (*ii*) *scheduling service call invocations*, to exploit parallelism and thereby minimize the makespan. Clearly, these issues are inter-related; efficiently assigning service executions to peers depends on balancing their load, and vice-versa. We use the term *AXML planning* in a general sense, to indicate both tasks (*i*) and (*ii*).

AXML planning is essentially characterized by scheduling a complex DAG to heterogeneous machines, which is an NP-complete problem [15, 33]. Because of such a complexity bound, we focus our optimization analysis on *finding suboptimal solutions in reasonable time*. Furthermore, as discussed in Section 4.4, intensional answers changes the specification of a materialization problem at runtime. Since usually peers cannot foresee these results, we assume AXML planning is initially restricted to the dependency graph obtained before the materialization starts, and occasional intensional answers trigger some re-optimization procedures. However, in our approach, such a re-optimization is localized to specific subgraphs whenever this is possible. We address this problem in Section 6.

Search space of materialization alternatives. Thus far we have considered materialization phases as totally-ordered sequences of service call invocations. Nonetheless, the dependency graph of an AXML document does not impose a total order on these invocations. Hence, many different invocation sequences can be used to materialize a document. For example, in the dependency graph of Figure 8, we may have invocation sequences starting with sc2, sc4, sc6, sc8 or sc10. Also, some service calls may be invoked in parallel. In our example, for instance, nodes sc4 and sc6 can be invoked independently.

There are several techniques to allocate resources from a pool of available machines for job scheduling [15, 18, 33, 55]. However, planning resource selection for AXML materialization is a different problem due to several reasons. In particular, in an AXML document each service call node has its own execution and delegation scopes, and a good resource configuration has to conciliate the diverse execution and data flows possibilities in a highly-dynamic scenario. Based on the combinations of callers and executors of each service call, the number of possible configurations of a de-

pendency graph Δ is given by:

$$\#locationConfigs(\Delta) = \prod_{x=1..|V^*|} |L_x^E| \times (|L_x^C| + 1) \quad , \quad (3)$$

where V^* is the set of all nodes in V plus the new node instances triggered by collateral calls. Now, if we consider also the possible invocation sequences, the number of AXML materialization alternatives is bounded to:

$$\#plans(\Delta) = |V^*|! \times \#locationConfigs(\Delta) \quad . \quad (4)$$

In the worst case, when any peer is both a provider and a caller candidate for every service call node, this number becomes $|V^*|! \times n^{2|V^*|}$, where n is the number of distinct peers. Even for simple scenarios, $\#plans(\Delta)$ tends to be large due to the exponential nature of the problem.

To reason about these possibilities, we introduce a central element of AXML planning: a *materialization plan*, which is derived from the dependency graph of an AXML document. However, instead of a complex graph, we use trees to represent a plan. More precisely, we consider the *minimum forest of spanning trees* of a dependency graph; namely, the minimum set of trees containing all the nodes and edges of the graph. Also, in a materialization plan, these tree nodes are labelled by operators of an algebra, which represent adequate materialization alternatives. We use the symbol $\mathcal{A}$ to indicate a finite set of algebraic operators. In Section 6, we present a formal definition for the minimum forest of spanning trees of an AXML document, and describe how such a forest is generated and converted into materialization plans. It should be noted that a tree-based representation is interesting for several reasons, specially because it enables the optimizer to reduce the complexity of AXML planning by partitioning the problem into simpler and possibly independent tasks (using the Divide&Conquer heuristic [44]). Basically, AXML planning is encoded in the following structures.

DEFINITION 23 *Given a service call node v held by peer P_v, an* invocation plan IP_v *is an expression $\langle P^E, P^C \rangle$, where P^E and P^C are peers that* can *invoke and execute v, respectively, such that $P^E \in L_v^E$ and $P^C \in (L_v^C \cup P_v)$. The term $\widehat{IP_v}$ denotes the set of all possible invocation plans of v, according to L_v^C and L_v^E.*

DEFINITION 24 *Let Δ be a dependency graph. A* materialization plan $\mathcal{M}$ *for Δ is of the form $\langle \Lambda, \mathcal{O}, \mathcal{L}, \succ, P_m \rangle$, where:*

- *Λ is the minimum forest of spanning trees of Δ;*
- *$\mathcal{O}$ is a labelling function that associates every node in Λ with an operator in $\mathcal{A}$;*
- *$\mathcal{L}$ is a mapping from each node v in Λ to invocation plans in $\widehat{IP_v}$;*
- *$\succ$ associates with each node in Λ a total order on its children; and*
- *P_m is the* master peer *of $\mathcal{M}$, namely the peer that holds $\mathcal{M}$ and where its persistent results must arrive.*

We say that $\mathcal{L}$ and $\succ$ are, respectively, the location scope *and the* invocation schedule *of $\mathcal{M}$. Moreover, $\mathcal{M}$ is* physical *if both $\mathcal{L}$ is total and it maps each node in Λ to exactly one invocation plan; otherwise, $\mathcal{M}$ is* abstract.

In a materialization plan, the *height* of a node indicates the size of the longest path from the node to a leaf node. This property can be: *simple* (denoted by *h*), if it considers only simple edges; *collateral* (*ch*), when it is based on paths that start with a collateral edge of the node and that may include transitive collateral calls; or *absolute* (*ah*), which is the highest value between *h* and *ch*. A variation of this property is the *least height* of a node, which is the size of the shortest path from the node to a leaf. We prefix the height with "*l*" to indicate such a variation (*e.g.*, *lh* denotes the least simple height). We assume the size of a path is given by the number of nodes on it, including the origin and the destination, and leaf nodes of the plan have $h = 1$. Unless stated otherwise, hereafter we refer to the simple height of the nodes by default.

Observe that the evaluation of a physical plan corresponds to a grounded invocation track of a materialization phase. This correspondence is important because it enables the optimizer to control the plan evolution, namely to correctly make the necessary updates to the plan during its evaluation. It is also worth noting that one can determine a materialization plan for some subset of service call nodes of a document, based on the corresponding dependency graph. In this case, we have a *subplan* of the document.

Performance metric outline. Comparing alternative materialization plans requires estimating their makespan. To calculate this metric, it is necessary to consider the sequential computations of the plan, as well as its peer assignment load. In workflow systems, the makespan of a physical plan is typically estimated by its *critical path* [18, 33, 63], namely the path of sequential executions with the larger completion time. Similarly, in AXML planning, sequential executions are mostly determined by invocation constraints. We can estimate the *static critical path* of each spanning tree of a physical plan by calculating the costs of both service execution and data transfers of its nodes (which is done recursively, from the leaves). Such a path is said "static" because the load of service executions of the involved peers is disregarded. On the other hand, the *dynamic critical path* accounts the sequential processing of service executions assigned to each peer.

We describe the basic formula to estimate the costs of Web service invocations in [44], which are compounded with other costs, such as delegation costs, in the model

presented in Section 7. We consider both execution and communication costs may be weighted, to calibrate the cost model according to the *computation-to-communication ratio* (CCR) [33] of the AXML setting.

An issue in estimating the makespan of a materialization plan is that *delegating AXML materialization implies in a non-deterministic execution scenario*. Due to P2P collaboration, it is impossible to know the exact position of the service call invocations of an AXML document on the execution queue of each peer involved in the materialization process. Consequently, we cannot determine precisely the dynamic critical path of a dependency graph. This represents an important restriction on the invocation scheduling problem, since it prevents the direct use of current algorithms to schedule workflows tasks onto heterogeneous machines, such as proposed in [15, 18, 33, 46, 47, 63, 66]. These algorithms are usually based on estimates such as the "*earliest start time*" and the "*latest start time*" of a task on a machine, which cannot be obtained with exactitude in AXML settings. The main reason is that peers are not dedicated resources and some parts of the dependency graph may be evaluated in parallel without global synchronization of service invocations. Nevertheless, ignoring current invocation assignments during the optimization may yield schedules similar to those based on the "*Minimum Execution Time*" (MET) heuristic [18], which may cause significant load imbalance across peers. For example, if a peer outperforms the others in many services, a load-blind optimizer may exhaust such a peer with excessive assignments. Worst, this would also stretch the makespan with massive sequential processing. To take into account service assignment load during AXML planning, our cost model (presented in Section 7) defines the *energy factor* of a peer in a materialization plan, which is used to dynamically weight the costs of plan nodes.

Optimization strategies. Many different strategies can be used to generate and compare alternative materialization plans (based on makespan estimates) of an AXML document. Each strategy is determined by an algorithm to produce these plans, and by a method used to compute their makespan, as follows.

DEFINITION 25 *Let MS be an objective function that estimates the makespan of materialization plans. An* optimization strategy *is an expression $\langle \Upsilon, MS, \mathsf{ceil} \rangle$, where Υ is an algorithm that finds a plan $\mathcal{M}$ for a given dependency graph Δ, and* ceil *is a performance ceiling such that $MS(\mathcal{M}) \leq \mathsf{ceil}$. The optimization algorithm Υ is* optimal *if $MS(\mathcal{M})$ is minimal for any Δ and for all possible plans of Δ. Moreover, we say that Υ is* complete *if $\mathcal{M}$ is physical whenever such a plan exists for Δ.*

A straightforward optimal strategy to find materialization plans is to adopt an exhaustive search, which yields all possible combinations of callers and executors, and of invocation sequences. However, such a strategy is often unfeasible due to its high time complexity (see Equations 3 and 4), and heuristic approaches are necessary to prune the search space. For instance, the optimizer can use the *Divide&Conquer* (D&C) heuristic to split the problem into smaller disjoint parts [44], which correspond to the connected subgraphs of the dependency graph. Thereby, the total complexity is reduced from a product to a sum of the complexity of the problem parts. Suppose there are n connected subgraphs in a dependency graphΔ, then the number of alternative physical plans is given by:

$$\#plans^{D\&C}(\Delta) = \sum_{i=1..n} \#plans(\Delta_i) \quad , \qquad (5)$$

where $\#plans(\Delta_i)$ denotes the number of alternative plans of each subgraph Δ_i. Since each subgraph is evaluated independently, the D&C strategy is optimal only if there is no interference between the enactment of the subgraphs. Namely, if the corresponding materialization plans involve distinct location scopes. Otherwise, some optimization decisions of a materialization plan might disregard the performance penalties resulting from the evaluation of other plans. Nonetheless, the main problem of the D&C strategy is that shared dependencies and collateral calls often imply graphs with a few, yet complex connected subgraphs. Hence, in these cases, the optimizer cannot explore significant complexity reductions. In Section 6, we propose an algorithm that maximizes the number of subgraphs of an AXML document by virtually replicating shared nodes.

Another class of heuristic strategies consists in setting arbitrary plan configurations. For example, the optimizer can systematically: set the master peer as the caller of the service invocations; and choose each service executor from the best provider w.r.t. execution time (namely, using the MET heuristic). We call this particular strategy MMET, referring to "*Master caller and MET heuristic*". In this case, the number of plan configurations that are analyzed for a dependency graph Δ is bound to:

$$\#locationConfigs^{MMET}(\Delta) = \sum_{x=1..|V^*|} |L_x^E| \quad . \qquad (6)$$

Although the MMET strategy may save significant optimization time, it does not explore P2P collaboration. Moreover, the MET heuristic ignores the invocation scheduling, and usually produces poor makespans [18]. Notice the MMET strategy is clearly suboptimal.

An important aspect of AXML planning is that, in general, *cost functions for makespan estimation are not monotonic*, in the sense defined in [25]. Namely, optimizing a subproblem may increase the overall makespan, as well as increasing the cost of a subproblem may lead to a better overall solution. For that reason, materialization plans need to be entirely specified to be properly compared. However, AXML settings are highly-dynamic scenarios, thus generating complete materialization plans may produce solutions

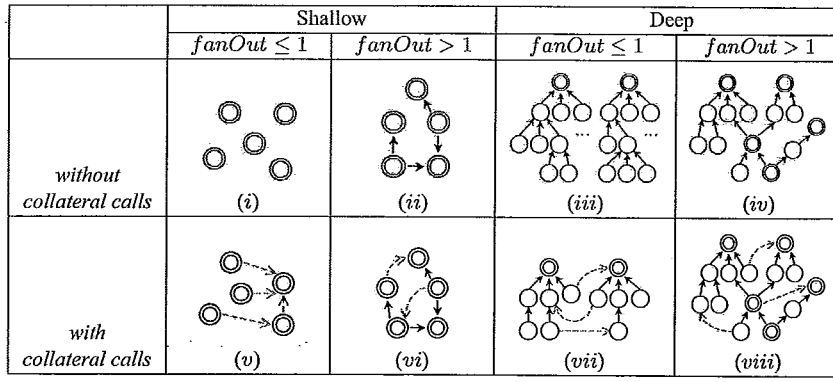


Figure 12. Main topologies of dependency graphs.

that are invalid at runtime. We propose an optimization strategy that interleaves AXML planning and materialization. We first identify independent tasks (spanning trees) of the graph, and then evaluate them separately. For each tree, we incrementally generate the corresponding plan based on the topological order of the service invocation nodes. The plan evaluation algorithm basically consists in climbing up a (partial) plan, from the leaf nodes to the root, considering subtrees of fixed heights. For each subtree, we analyze its materialization possibilities and generate a subplan, which is executed before resuming the AXML planning and materialization. It is worth mentioning that our strategy is based on a meta-heuristic that can be combined with other approaches to find efficient solutions for the subplans.

5.5 Main Problem Topologies

The shape of a dependency graph determines most of the opportunities for P2P collaboration, as well as the requirements of scheduling service invocations. To get a better understanding of the issues behind AXML materialization, we analyze the main graph topologies. We highlight three dimensions: (1) *the length of paths of sequential data transfers with temporary nodes*, which is related to the paths of invocation dependencies on the graph; (2) *the shared dependencies of the graph*, namely the nodes with $fanOut > 1$; and (3) *the sequential collateral invocations* of service call nodes. We say that a dependency graph is *deep* if it has long paths of invocation dependencies with few persistent nodes, and *shallow* otherwise.

Based on these criteria, we have the graph topologies of Figure 12. Considering shallow graphs, the simplest topology is an empty dependency graph, shown in item (*i*); it consists essentially of a *bag-of-tasks*, without data transfers between service call nodes. Optimizing such a graph concerns mostly load balancing. A variant of this topology including collateral calls is illustrated by item (*v*), where the optimizer has to schedule parallel sequences of service invocations without data transfers between service executions. Also in this case, service execution costs are the performance yardstick. Notice that with shared dependencies and/or some data flows between service invocations (*i.e.*, topologies (*ii*) and (*vi*)), the optimization goal remains load balancing, since only persistent nodes are handled. Yet, if persistent nodes do not represent large data transfers, then AXML delegation can be explored to improve parallelism in shallow graphs, thereby reducing their makespan.

Observe that shallow topologies are not expected to be frequent in AXML documents, because they do not encompass intensional parameters. Another class of problems is represented by deep graphs, which are more usual in AXML materialization. For these graphs, the optimizer can try to find fast links between the participating peers in order to reduce communication costs related to temporary nodes. The typical case is topology (*iii*), where the exit points of the graph denote independent materialization tasks with few required data transfers to the master peer. Interestingly, these tasks remain independent in topology (*vii*), despite the collateral edges connecting them. The reason is that collateral calls represent new instances of service invocations, which are "*clones*" of the referred nodes. On the other hand, the shared dependencies in topology (*iv*) determine some synchronization points between materizalization tasks, which are not independent from each other.

The first attempt of our optimization strategy is to simplify the dependency graph by transforming it into a forest of (possibly deep) independent tasks. The resulting graph is close to the topology in (*iii*), which can be more easily evaluated by the optimizer. Such a topology favours parallel execution, along with the reduction of communication costs. Next, we present the proposed optimization strategy.

6 Optimizing AXML Materialization

Currently, the ActiveXML platform lacks a performance-oriented approach to the optimization of AXML documents with abstract service references. Basically, ActiveXML peers support only one materialization strategy, which is uniquely determined by explicit service call attributes and (possibly) typing control. In this Section, we propose a cost-based optimization strategy to improve AXML materialization by dynamically analyzing alternative materialization plans. We describe the overall optimization strategy in Section 6.1, and its main steps are detailed as follows. Section 6.2 presents techniques based on spanning trees to generate initial materialization plans from arbitrarily-complex dependency graphs. These plans are encoded with an algebra that enables the optimizer to perform incremental and collaborative AXML materialization. In Section 6.3, we outline an algorithm to partition a materialization plan into tasks, and a priority-based mechanism to order these tasks. In particular, we explore both the inter-task and the intra-task parallelism degree of a plan to sort its tasks. Going further, we propose in Section 6.4 an algorithm that scans a materialization plan in topological order and dynamically generates "good" physical plans based on cost metrics. Finally, we discuss task delegation and collaborative optimization in Section 6.5.

The proposed strategy is dynamic in the sense it partially materializes the plan before completing the optimization. Moreover, resource allocation and planning are performed at runtime. It is also adaptive, as defined in [28], since it makes the optimizer choices sensitive to changes in the P2P system at each step of the materialization process.

6.1 Dynamic Optimization Strategy

To materialize an AXML document, the optimizer has to deal with two major issues: a *huge search space* of materialization alternatives, and the *unpredictability* of the P2P setting. In a static approach to generate materialization plans, all the service calls, the interactions among them, their service providers, and communication costs are assumed to be known before the optimization starts. Clearly, this approach is not suitable for AXML materialization. Ideally, the optimizer should react to changes in the environment, and this should not be based solely on plan reoptimization, which is often quite expensive in an unstable setting. We propose an optimization strategy that exploits dynamic techniques to reduce complexity and to enable the system to adapt to both system performance and membership fluctuations. In our approach, materialization plans are not produced at once, and reoptimization is triggered only when really necessary. Not surprisingly, our techniques mostly rely on *splitting the dependency graph into smaller pieces*. However, the main question here is how to partition the materialization problem such that important performance aspects, such as data flows between service invocations, are preserved and considered by the optimizer.

```
1  procedure DynamicOptimize(Δ, k)
2  {Efficiently materialize Δ based on dynamic plan
     generation of k-depth steps.}
3  begin
4    Generate an initial abstract plan M_i from Δ
5    Compute the set T_i of materialization tasks of M_i
6    Order the tasks of T_i by priority level
7    for each task t in T_i do
8      if t is to be delegated then
10       Pick a new master peer P'_m in N for t
11       Delegate t to P'_m
12       go to next task
13     end if
14     Split t into k-depth subplans
15     for each subplan M_x in t in topological order do
16       Locate providers and executors for M_x
17       Generate alternative physical plans of M_x
18       Rank physical plans and pick the best M_best
19       Execute M_best
20     end for
21     Re-evaluate the order of tasks in T_i
22   end for
23 end
```

Figure 13. Overall optimization algorithm.

Inspired on Web protocols, which present results as they arrive (instead of waiting for complete documents), our optimization strategy also allows to minimize the time to obtain the *first results* of the document materialization. We *interleave materialization planning and execution*, thus the peer optimizer can decide how to proceed after partial executions, when it may have more up-to-date information on the system status.

The basis of our optimization strategy is to work on the materialization of an AXML document by using its dependency graph, which explicitly shows all the invocation constraints of the embedded service calls. The optimizer unfolds this graph into a simplified tree-based structure, and then produces an *initial abstract plan*, whose location scope and invocation schedule are not determined. Such a plan is partitioned into materialization tasks, which can be further split into subplans of height k according to the topological order of the service call nodes. For each subplan, the optimizer annotates the execution and delegation scopes of its nodes, and then generates its alternative physical subplans. These equivalent subplans are then ranked based on some cost metrics, and the "best" (but not necessarily optimal) alternative is picked and evaluated. This process is repeated until all the tasks and their subplans are evaluated. Optionally, the optimizer can delegate some tasks to be completely evaluated by other peers. The overall algorithm of our op-

timization strategy is described in Figure 13. In the following, we detail these optimization steps.

6.2 Generating Materialization Plans

To generate a materialization plan, the optimizer has to associate the service call nodes of the dependency graph with adequate evaluation operators, which will actually process each service request. For example, a service call may be invoked locally (*i.e.*, by the master peer) or delegated to another peer; each of these cases require a different operator to handle the service call node. These operators are interpreted by the optimizer, and their service call results are inserted into the AXML document. Nevertheless, a dependency graph is rather a complex structure to be directly processed by the optimizer, due mostly to shared dependencies and collateral calls. Therefore, we first encode the dependency graph using a simpler, tree-based structure that is more adequate for distributed evaluation. In this transformation process, we also attempt to apply a *Divider&Conquer* heuristic to reduce the problem complexity. In particular, we extract the *spanning trees* of the dependency graph, such that the optimizer can identify and evaluate independent tasks. Then, we use an algebra of materialization operators to generate an initial plan from these trees. Such a plan contains only *abstract operators* (which lack location scope information) and its purpose is to enable the optimizer to produce alternative physical plans.

Spanning trees. Several classical algorithms can be used to build spanning trees from an arbitrary graph. For instance, the well-known Prim's algorithm [11] has time complexity $O(|V|^2)$ using an adjacency list as graph structure, and $O(|V|log|V|+|E|)$ for a heap-based graph. This algorithm begins with a node of the graph as the current tree, and then builds its *border*, that is a set of all the nodes that can be reached from this start node. Each node in the border is added to the current spanning tree and expanded recursively. These steps are repeated until all the spanning trees of the graph are obtained. The roots of these trees are the seeds of the algorithm. In our case, only exit points of the dependency graph are used as seeds. Moreover, the border is built considering the opposite direction of simple edges, as stated next.

DEFINITION 26 *Let Δ be a dependency graph and v_x, v_y be two nodes in Δ. The node v_y is* reachable for spanning *from v_x iff either $v_y \rightarrow v_x$ or $v_x \hookrightarrow v_y$ is in Δ. Furthermore, the* border *of v_x consists of the set of all the nodes in Δ that are reachable for spanning from v_x.*

If the dependency graph has only connected subgraphs that represent trees, then we can easily identify and evaluate its independent tasks. However, an AXML document usually involves shared dependencies and/or collateral calls, which denote subgraphs that do not correspond to trees. To handle this, we have to build a flat representation of the dependency graph, where each node belongs to exactly one tree. This is done by *node detachment* – namely, by identifying and separating subgraphs that are connected by some service call (*i.e.*, which have service calls in common). We propose two special transformations to obtain such a representation:

- *node replication*, that is to replace the node by a set of exact copies of it, which represent the same instance of the corresponding service call. This transformation is applied to separate subgraphs connected by shared dependencies; and
- *node cloning*, which consists of adding new instances of a service call node to the dependency graph. It is used to represent collateral calls.

Figure 14 shows the algorithm used to flatten a dependency graph based on these two transformations. Basically, each shared dependency (namely, a node with $fanOut > 1$) is replaced by $fanOut$ replicated nodes, such that each replica inherits exactly one of the outgoing simple edges of the original node, and all of its incoming simple edges. For nodes that are pointed as collateral calls, we clone their entire subtrees for each incoming collateral edge. Replicating shared nodes has time complexity $O(\otimes \times |E|)$, while the bound for node cloning depends on the algorithm used to unfold spanning trees.

From a flat dependency graph, we can expand the spanning tree of each exit point based on the border criteria of Definition 26. The result of this process consists of the following set of (possibly related) trees.

DEFINITION 27 *Let Δ be a dependency graph, and V^{exit} the set of exit points of Δ, where $V^{exit} \subseteq V$. The expression $\Lambda = \langle ST, \varrho, \hookrightarrow \rangle$ represents the* minimum forest of spanning trees *(or* MFST, *for short) of Δ, where ST is a set of unordered trees of Λ, such that:*

- *shared nodes of Δ are properly replicated in ST;*
- *collateral calls of Δ are properly cloned in ST; and*
- *for each node v_x in V^{exit} there is a tree st_x in ST that is rooted by v_x and which results from recursively expanding the border of v_x.*

Furthermore, the function ϱ associates replicated nodes in Λ with their original node in Δ, and $\hookrightarrow$ denotes a distinguished subset of edges in Λ, which correspond to the outgoing collateral edges of Δ. The number of spanning trees of Λ is denoted by $|\Lambda|$.

Observe that ST is equivalent to Δ_{flat}. We assume that tree nodes keep their properties from the dependency graph, such as the persistency flag and invocation status. Also, since the edges of a spanning tree are not directed, collateral

```
1  function FlattenDependencyGraph(Δ): Δ^flat
2  {Transform shared nodes (due to shared dependencies
     and collateral calls) to disconnect subgraphs of Δ.}
3  begin
4    let Δ^flat = Δ
5    for each node v in ⊗^flat
6      if fanOut(v) > 1 then {shared dependency}
7        Replicate(v, Δ^flat)
8      end if
9    end for
10   for each edge v_x ↪ v_y in E^flat (↪) do   {collateral call}
11     Clone(v_y, v_x ↪ v_y, Δ^flat)
12   end for
13   return Δ^flat
14 end

15 procedure Replicate(v, Δ)
16 {Replicate node v in dependency graph Δ
     according to its outgoing simple edges.}
17 begin
18   for each v → v_x in E do
19     let v_rep be a new replica of v   {an exact copy of v}
20     Add v_rep to V
21     Add v_rep → v_x to E
22     for each v_y → v in E do
23       Add v_y → v_rep to E
24     end for
25     for each v ↪ v_z in E (↪) do
26       Add v_rep ↪ v_z to E (↪)
27     end for
28     Delete v → v_x from E
29   end for
30 end

31 procedure Clone(v_y, v_x ↪ v_y, Δ)
32 {Clone subtree rooted at node v_y through the
     collateral call v_x ↪ v_y in graph Δ.}
33 begin
34   let v_clone be a new clone of v_y   {a new instance of v_y}
35   Add v_x ↪ v_clone to E (↪)
36   Unfold the spanning tree rooted at v_y into v_clone
37   Delete v_x ↪ v_y from E (↪)
38   let IN be the set of incoming collateral edges of v_y
39   if |IN| = 0 then
40     Delete v_y from Δ
41   end if
42 end
```

Figure 14. Algorithm to flatten (by node replication and/or cloning) a dependency graph.

edges require proper distinction for their particular "*fire after*" semantics. In fact, we consider that collateral calls are not expanded as regular child nodes in the spanning trees, but as *annotations* on the service call nodes.

DEFINITION 28 *Given a MFST* Λ *and two nodes* v_y *and* v_z *in* Λ, *we say that* v_y *is a* collateral annotation *on* v_z *iff* $v_z \hookrightarrow v_y$ *in* Λ.

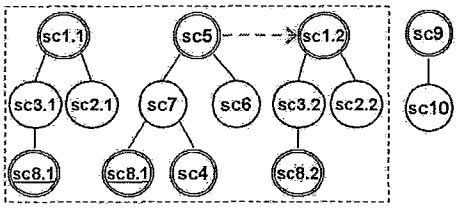


Figure 15. MFST of the *SwapWorkspace* graph.

Figure 15 shows the MFST obtained from the graph of Figure 8; replicated nodes have underlined text, and collateral annotations are denoted by dotted arrows. Cloned nodes have distinct IDs, which we represent by "scX.Y", where X is the ID of the original node, and Y is the specific ID of the clone. Observe that the spanning trees of a dependency graph may be grouped into (possibly overlay) clusters, such that each cluster has all the trees with node replicas of a service call. More precisely, a cluster represents some connected subgraph of the dependency graph. For example, there is one cluster in the MFST of Figure 15, which is indicated by a dotted rectangle. Although there are not precendence constraints between the trees of a cluster, they are not independent from each other: replicated nodes express synchronization points in the evaluation of their respective spanning trees. Yet, the trees of a cluster become independent once one of the replicas is evaluated. We address these issues in Section 6.3.

Algebra of materialization operators. Having computed the MFST of a dependency graph, the optimizer has to turn the resulting trees into a materialization plan. Basically, this can be done by replacing tree nodes in the MFST by operators of an algebra $\mathcal{A}$, whose main requirements are the support for Web services invocation and for P2P collaboration. Also, such an algebra should allow the optimizer to incrementally evaluate a materialization plan. Based on these requirements, we propose the algebra described in Table 2. We distinguish three groups of operators:

(*i*) the *abstract operators* μ and ρ, which represent the possible combinations of executors and callers of service call nodes in a plan. These operators cannot be interpreted as service executions since they lack specific invocation details, such as the Web service endpoint. For example, the μ operator has to be converted into some physical operator (*e.g.*, **invoke**) in order to result into a service invocation;

(*ii*) the *physical operators* **invoke**, **fetch** and δ, which contain all the information required to invoke a Web service. Observe that δ does not point directly to services that are requested in the AXML document. Instead, it represents the invocation of a basic Web service for P2P collaboration, which is going to handle

Table 2. Algebra of operators for dynamic and decentralized AXML materialization.

Operator	Description
$\mu(v)$	The ***materialize*** operator tells the optimizer to determine an invocation plan for the service call node v. Namely, to choose both a caller and an executor for v among the peers in its location scope (L_v^C and L_v^E, respectively). *Pre-conditions*: L_v^E is not empty; none of its descendant nodes in the plan is an auxiliary operator; and every descendant node has at least one provider in its execution scope.
$\rho(v)$	The ***retrieve*** operator is slightly different from $\mu(v)$. Additionally, it informs the optimizer that v is a shared dependency. That is, it behaves like $\mu(v)$ with the additional cache operation for future retrievals, unless v is already evaluated. Otherwise, it retrieves the invocation plan of v from the cache. *Pre-conditions*: either there is an entry for v in the cache or the same pre-conditions as for $\mu(v)$ hold.
invoke(v, IP_v)	This operator is interpreted during evaluation as an invocation of v from peer P^C to execute the requested Web service at peer P^E, such that $IP_v = \langle P^E, P^C \rangle$ is an invocation plan of v. *Pre-conditions*: all the dependencies of v are inactive.
fetch(v)	It informs the optimizer to look for previous invocation results of v at the system cache before materializing it. It corresponds to a physical version of ρ. In particular, it behaves as **invoke** with the caching feature. *Pre-conditions*: either there is a cache entry for the invocation result of v (if v is not inactive), or there is an invocation plan for v in the cache and all the dependencies of v are inactive.
$\delta(v, IP_v)$	The ***delegate*** operator asks peer P^C, from the invocation plan $IP_v = \langle P^E, P^C \rangle$ such that $P^C \in L_v^C$, to materialize (possibly with some further optimization) the subplan rooted at v, by solving itself all the necessary intensional parameters and collateral calls. *Pre-conditions*: all the δ operators in its subplan, as well as all the **invoke** operators whose caller is the current master peer, are evaluated. Also, P^C must be a neighbor (that is, $P^C \in \mathcal{N}$).
$\Theta(v)$	The ***optimize*** operator denotes a request for a remote peer (*i.e.*, neighbor) to optimize the subplan rooted at v, possibly including its materialization. *Pre-condition*: at least one peer in $\mathcal{N}$ supports Θ.
locate(v)	This auxiliary operator denotes a request for a neighbor to discover the execution scope of both the Web service of v and the services of all the μ operators in the subtree of v that have an empty execution scope. *Pre-condition*: at least one peer in $\mathcal{N}$ supports **locate**.
pipe	It is used to simplify updating the dependency graph with intensional answers, as explained in Section 4.4. This operator represents a pipeline that gathers the results of its children and transmits them to its parent node in the plan. Its Web service may be executed either locally or at some peer in $\mathcal{N}$. *Pre-condition*: all of its children are ready.

one or more service requests of the document; and

(*iii*) the *auxiliary operators* Θ, **locate** and **pipe**, which are mainly used to decentralize the optimization process. Similarly to δ, these operators point to some basic collaboration service. Initially, they may lack the target peer, but the optimizer must set them to some specific neighbor (or locally, in case of **pipe**) before their evaluation.

Each algebraic operator in $\mathcal{A}$ is associated to a specific set of actions, according to its goal. To be evaluated by the optimizer (thus triggering its actions), the operator has to satisfy some pre-conditions, as described in Table 2. The semantics of evaluating an operator varies according to its type. In general, the evaluation of abstract operators does not trigger any service invocation, but only makes the optimizer to analyze their alternative physical plans in order to choose the "best" options. Such an analysis usually results in replacing the abstract operators by their physical counterparts. For auxiliary operators, evaluating them means choosing a peer (neighbor) and then invoking the corresponding Web service for P2P collaboration. Observe that both abstract and auxiliary operators do not cause any changes to the AXML document, except for Θ when it includes plan materialization (that is, the subplan evaluation is completely delegated to other peer). On the other hand, evaluating physical operators results in invoking some service calls of the AXML document and updating its contents.

The optimizer uses these operators to compose materialization plans as follows. First, it generates an initial plan with abstract operators. Then, this plan is successively transformed by replacing, adding and/or consuming operators. Plan transformations may be due to either operators evaluation or traditional rule-based optimization. Table 3 enumerates the possible transformations obtained by evaluating algebraic operators. Notice all physical operators are either consumed or replaced by an intensional answer. Also,

Table 3. Evaluation of algebraic operators.

Original operator	Resulting operator(s)
$\mu(v)$	either **invoke**(v, IP_v) or $\delta(v, IP_v)$
$\rho(v)$	**fetch**(v) | *cached plan of* v
invoke(v, IP_v)	*none* | *intensional answer*
fetch(v)	*none* | *intensional answer*
$\delta(v, IP_v)$	*none* | *intensional answer*
$\Theta(v)$	*subplan for* v | *none* | *intens. answer*
locate(v)	*subplan rooted with* $\mu(v)$
pipe	*none*

the Θ operator may be consumed if it includes materializing its delegated subplan. Furthermore, we assume the optimizer can apply the following basic transformation rules:

1. it inserts Θ operators to partition the children of a node if they are too numerous;
2. it replaces a μ operator by **locate** if the subplan of μ has at least some pre-defined percentage of nodes missing the execution scope (if the peer failed to identify the providers of these nodes); and
3. it replaces an **invoke** operator by δ if its caller (*i.e.*, P^C) is neither the master peer nor the caller of its parent node.

These rules are meant to be explored at the discretion of the optimizer, in different phases of the optimization strategy. For example, rule 1 is used in early optimization phases to break the plan into pieces of reasonable size, while rule 3 is used to determine delegation points when generating physical plans. Rule 2 is related to *contingency planning*, which we discuss in Section 6.4.

It is worth mentioning that our algebraic approach is extensible, since one can add other operators to $\mathcal{A}$, as well as new rules to handle these operators. For example, the **invoke** operator could be further specialized into other physical operators, such as a synchronous and an asynchronous operator (for continuous Web services).

Generating initial plans. Although the MFST is a canonical representation, usually there are many different plan alternatives for its trees, according to the algebraic operators used to handle the service requests. Nevertheless, the optimizer can rely on abstract operators to rather perform a simple (and fast!) analysis to generate initial plans. Such an analysis assumes that each service call node yields either a μ or a ρ operator. (This last operator is used if the node is a replica.) The initial plan is called *abstract* because its nodes consist essentially of abstract operators.

The *GenerateInitialPlan* algorithm of Figure 16 shows the steps to generate an initial abstract plan from a dependency graph. After computing the MFST, an initial plan is set as a carbon copy of if. Then, for each tree, nodes are labelled by abstract operators accordingly. Once all the nodes

```
1  function GenerateInitialPlan(Δ): M_i
2  begin
3    FlattenDependencyGraph(Δ)
4    Compute the MFST Λ of Δ
5    let M_i such that Λ_i = Λ
6    {Determine the labelling function O_i}
7    for each tree st in Λ_i do
8      for each node v in st do
9        if v is a replicated node then
10         Replace v by ρ(v)
11       else Replace v by μ(v)
12       end if
13     end for
14     if the master peer cannot evaluate st then
15       let v_root be the root of st
16       Replace v_root by Θ(v_root)
17     end if
18   end for
19   {Both L_i and ≻_i are left undetermined for now.}
20   return M_i
21 end
```

Figure 16. Algorithm to generate initial plans.

of a tree are associated with algebraic operators, the optimizer decides (at 14) whether or not it is going to evaluate the respective subplan. Notice the optimizer may choose to delegate a subplan in many cases; for the initial plan, we consider this happens when either the MFST has too many trees or the master peer is overloaded. Also, for simplicity, we assume delegated subplans do not contain replicated nodes (to avoid problems due to data coupling). At this phase, we neither set an invocation schedule ($\succ$) nor determine the location scope ($\mathcal{L}$) for the abstract plan, which will be progressively defined during the optimization process.

Figure 17 depicts the initial abstract plan for the dependency graph of Figure 8. Each node represents an algebraic operator, which is specified in the node label. Nodes may also have collateral annotations (the "*cp*" reference under the label, in Figure 17). Additionally, the optimizer may annotate nodes with some supportive information, such as node height. Persistent nodes are denoted by double-line rectangles. We divide the materialization plan into three areas. The central area is the "*main plan*", which consists of all the spanning trees of the materialization plan, such that subplans rooted by replicated nodes are represented by either a ρ or a **fetch** operator. The subplans of replicated nodes are kept in the "*cached plans*" area, and the "*cp*" references point to subplans in the "*collateral plans*" area. The master peer of our example is P_1, according to Figure 1.

If the dependency graph does not change between subsequent materialization requests (or if changes are not significant), then the optimizer can store the initial abstract plan of an AXML document for further reuse. Also, the optimizer can propagate occasional updates on the graph to the

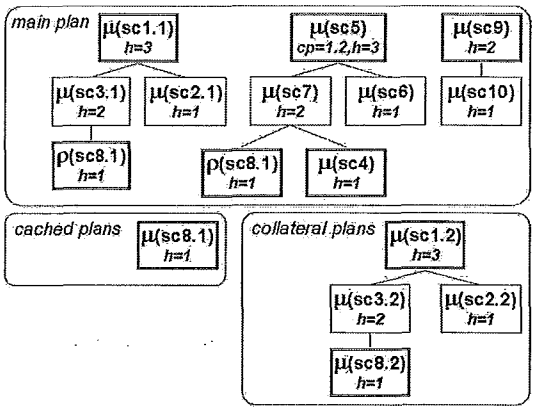


Figure 17. Initial abstract plan for the *Swap-Workspace* document.

plan according to the rules presented in Section 4.4, possibly by using **pipe** operators. For many changes, however, re-computing the initial plan from scratch may be less expensive than applying the respective updates.

6.3 Determining Materialization Tasks

In our optimization strategy, instead of processing the entire initial plan at once, the optimizer partitions it: first, into *materialization tasks*, and then into *k-depth subplans*. By materialization task, we consider a well-defined, self-contained goal of a materialization plan. To determine the tasks of a plan, we adopt an approach that focuses on its exit points, which correspond to the roots of the MFST. Also, for preliminary scheduling purpose, each task is associated with some evaluation priority, as defined next. Let $\mathcal{T}$ be a finite set of tasks identifiers.

DEFINITION 29 *Given a plan $\mathcal{M}$, the expression $\langle T, \pi \rangle$ denotes the set of* materialization tasks *of $\mathcal{M}$, where T is an injection of trees in $\mathcal{M}$ into tasks in $\mathcal{T}$, and π associates each task t of T with a* priority level.

We consider that each tree of a materialization plan yields a task. Our choice for this criterion is motivated by two main reasons. Following Proposition 2, we know the materialization process converges at the exit points of a plan after a finite number of service invocations (assuming a snapshot semantics). Therefore, these nodes enable the optimizer to use an objective function to properly estimate the makespan of each task. Besides that, exit points are always first-level service calls, and they constitute the ultimate contents of the document materialization.

Clustering tasks. Due to shared dependencies (*i.e.*, nodes labelled by ρ or **fetch** operators), the trees of a materialization plan are not necessarily independent. That is, plan

```
1  function IdentifyTaskClusters(M): Clusters
2  {Returns a collection of clusters indexed by shared
     nodes (or root nodes, for independent trees).}
3  begin
4    let Clusters = ∅
5    for each tree st in M do
6      let Rep be the set of replicated nodes of st
7      if Rep = ∅ then    {st is independent}
8        let v_root be the root of st
9        Create Clusters[v_root]
10       Add st to Clusters[v_root]
11     else   {st has some shared nodes}
12       for each node v in Rep do
13         let node v_origin such that ρ(v) = v_origin
14         if v_origin ∉ Clusters then
15           Create Clusters[v_origin]
16         end if
17         Add st to Clusters[v_origin]
18       end for
19     end if
20   end for
21   return Clusters
22 end
```

Figure 18. Algorithm to identify clusters of materialization tasks.

trees can be grouped into overlay clusters, where the trees of each cluster share some service call results. Identifying the clusters of a plan can be done by a simple algorithm, as shown in Figure 18. Such a data coupling usually restricts (or, at least, complicates) the distributed processing of a materialization plan. Yet, once a replicated node is evaluated and its result is passed to the other replica accordingly, its corresponding cluster stops existing. Determining independent tasks has many advantages. In particular, it enables parallel execution and the decentralization of the optimization process, which fit quite well in P2P systems. Moreover, partitioning the plan may significantly reduce its optimization complexity, as discussed in Section 5.4. Hence, to overcome the data-coupling problem, we try to find a tasks evaluation order that would gradually increase the *parallelism potential* of the plan, thus allowing the optimizer to effectively explore P2P collaboration. For simplicity, let us assume each independent task of a plan is a cluster. Thus, we can estimate the parallelism potential of a plan $\mathcal{M}$ as:

$$par_{\mathcal{M}} = |Clusters| \ . \qquad (7)$$

where $Clusters$ is the set of clusters of $\mathcal{M}$, including its independent tasks. Figure 19 describes the clusters found in the initial plan of Figure 17. Tasks in each cluster are represented by their root nodes. In this example, we have $par_{\mathcal{M}} = 2$. Observe this number may change after each task evaluation, according to the affected clusters.

$$Clusters[\text{sc8.1}] = \{\mu(\text{sc1.1}), \mu(\text{sc5})\}$$
$$Clusters[\text{sc9}] = \{\mu(\text{sc9})\}$$

Figure 19. Clusters of the *SwapWorkspace* plan.

Priority assignment. To rank the tasks of a materialization plan, we consider the optimizer assigns priorities for them. Such a metric can be determined based on (possibly a combination of) several different task properties, such as:

- the number of service calls;
- the abstract critical path;
- the absolute height of the root node;
- the expected makespan, based on previous execution times;
- the number related clusters; and
- the number of replicated nodes.

The first three criteria basically define the size of a task; the optimizer can be configured to evaluated tasks following either a "smallest-first" or a "biggest-first" policy, according to the system requirements. Observe that to estimate the size of a task, ideally the optimizer should consider the dynamic critical path of the plan. However, the plan operators are rather abstract at this phase of the optimization strategy, and this information is not available yet (trying to get it would be very costly). Besides these size-based criteria, more user-defined properties could also be used, such as the presence of certain Web services requests.

The last two bullets are related to parallelism potential. These clustering-based criteria reflect, respectively, the external and internal data coupling of a task. Notice that both size- and clustering-based criteria are important to efficiently schedule materialization tasks. Therefore, let t be a materialization task and $Clusters_t$ be the set of clusters that contain t. We compute the priority level π of t as:

$$\pi(t) = \pi_{size}(t) + \sum_{c \in Clusters_t} \pi_{cluster}(t, c) \ , \qquad (8)$$

where π_{size} is the size-based priority of t, and $\pi_{cluster}$ is the cluster priority of t (for each cluster c in $Clusters_t$). We assume that π_{size} is estimated by some arbitrary function, regarding some criteria such as those we enumerated. On the other hand, the cluster priority of a task is given by:

$$\pi_{cluster}(t, c) = \omega_{inter} \times \pi_{inter}(c) + \omega_{intra} \times \pi_{intra}(t, c) \ . \qquad (9)$$

The terms ω_{inter} and ω_{intra} denote weights to adjust parallelism priority, for inter-cluster and intra-cluster parallelism, respectively. These weights allow the optimizer to adapt priority assignment to different performance requirements and problem topologies. The *inter-task parallelism degree* π_{inter} of a cluster c indicates the potential of c for parallel evaluation (*i.e.*, if one of the tasks of c is evaluated), and it is given by:

$$\pi_{inter}(c) = |c| - 1 \ , \qquad (10)$$

where $|c|$ is the number of tasks on c. Analogously, the *intra-task parallelism degree* π_{intra} of a task t in a cluster c represents the branches of t that could be parallelized once c has been solved. The priority π_{intra} is directly given by the number of replicated nodes of c in t, since two replicated nodes cannot be in the same branch of a task.

Observe the optimizer starts the plan evaluation by distributing the tasks that were chosen to be optimized by other peers. Hence, we consider the highest priority level (denoted by π_∞) is assigned to these tasks by default. Recall that we assume these tasks do not belong to any cluster. Moreover, tasks with the same priority level can be further ordered by some deciding criterion, such as their expected makespan.

Blocking *versus* non-blocking tasks. In a regular peer (*i.e.*, with only one processor), tasks are usually evaluated in a *blocking mode*; namely, they are handled one-by-one, and each task blocks the evaluation of the others. In this case, using parallelism potential to compute priority can help the optimizer to explore tasks delegation. On the other hand, for parallel peers, the optimizer can directly explore the clusters of a plan to speedup its evaluation, since each task blocks the evaluation only within the scope of its clusters. It is worth mentioning this can be simulated with a multi-thread system in regular peers. Nevertheless, empirical results [37] have shown that managing several simultaneous connections to Web services usually penalizes performance. Notice that tasks delegation is a quite different technique, since the optimizer can use asynchronous Web services for collaboration, thus avoiding lasting open connections.

When parallelism is imperative, the optimizer may compute dynamically the priority levels of materialization tasks, since these parameters can significantly change after each task evaluation.

6.4 Dynamic Plan Generation

As we saw in Section 5.4, the search space of alternative materialization plans is dramatically large even for very small problem configurations. In general, exponential-complexity search problems cannot be solved for any but the smallest instances [45]. Also, breaking the optimization problem into pieces fosters P2P collaboration. Although materialization tasks are natural candidates as a plan splitting unit, usually they are not necessarily small enough to be efficiently optimized. Hence, we further split materialization tasks into subplans, which are used to finally generate and rank alternative physical plans.

Two main aspects rule generating physical plans in AXML optimization: the location scope ($\mathcal{L}$) of the requested Web services; and the invocation schedule ($\succ$) of the plan operators. Basically, the optimizer yields different combinations of service providers and callers (according to both the execution and delegation scopes of each service call node), along with different invocations sequences, to obtain alternative physical plans. Our optimization strategy adopts a variation of the *workflow-based generation approach* [15] to produce these plans. This is opposed to a task-based generation approach (which is quite popular in grid systems [24, 51]), where the optimizer makes greedy decisions for each plan operator. Notice a task in this context is just a plan operator. Typically, a greedy optimization algorithm processes a plan node-by-node, picking the best alternative for each node according to some heuristics and localized performance parameters. Its main advantages are the reduced complexity and the adaptability to changes in the system. Moreover, intermediate results can be shipped as soon as possible. However, in the AXML setting, there is a strong performance correlation between a plan node and its dependencies, and greedy decisions are usually very inefficient. In a workflow-based approach, the whole problem is considered to produce the search space. The analysis of complete plans enables the optimizer to reason about overall performance, but it is really inadequate for AXML materialization due to the exponential nature of the problem. We propose a new optimization strategy that combines advantages from these two approaches.

Dynamically producing k-depth subplans. The main idea of our strategy is to exploit a hybrid search technique, which performs a greedy analysis on *k-depth subtrees* of each materialization task. The k parameter determines the absolute height of each subplan that is analyzed by the optimizer. This parameter usually has a significant impact on the problem complexity, thus we assume k is a small integer (say, chosen from 2 to 4 inclusive). For each subtree, the optimizer generates and ranks alternative physical subplans, considering the relationships between the operators of the subtree (*e.g.*, data transfers and invocation sequencing). Once a good physical subplan is selected, it is executed before the optimization process is resumed. In Figure 13, the steps from line 14 to 20 describe the core of the proposed strategy.

Observe that, through our dynamic strategy, the optimizer can have a bird's eye perspective of height k of the materialization plan. This way, it can still be sensitive to overall performance issues, while keeping the plan complexity manageable. Moreover, the optimizer is able to adapt a plan to changes in the system, such as peers membership fluctuations. Also, it can handle incomplete problem specifications. For example, the optimizer does not have to know the execution scope of all the plan operators from the beginning of the optimization process. Instead, it can try to increase its knwoledge of the problem gradually, as the plan is evaluated. The k parameter can be either determined by the peer administrator or inferred (based on some heuristics) from the absolute height of the task root.

Computing split points. To partition a task, the optimizer has to compute its *split points*, namely the nodes that root the k-depth subplans of the task. The algorithm outline is shown in Figure 20. Loosely speaking, the optimizer walks the task in some arbitrary tree traversal (*e.g.*, in pre-order or in post-order); for each task operator, if its height is a multiple of k, then the optimizer sets it as a split point. The task root is considered a split point by definition. Notice we have to use the least height of the plan operators to properly split the task. The *mod* (for modulus) function in lines 33 and 34 of Figure 20 returns the remainder of an integer division.

Furthermore, two node occurrences require special treatment to determine the split points of a task. The first occurrence is of replicated nodes, which are essentially task leaves that point to some cached plans. When considering a replicated node for task splitting, either its respective cached plan is already evaluated or it is waiting for evaluation. In the last case, the optimizer can partition the corresponding cached plan similarly to a task, except that the root node of the cached plan (*i.e.*, the shared node) is not considered a split point unless its least height is a multiple of k. The intuition is that a replicated node has to be replaced by its cached plan. However, since a task may have several replicated nodes pointing to the same cached plan, the optimizer has to choose exactly which replica is going to be solved first (and be replaced by its cached plan). For replicated nodes in different subplans, this is done following the split points evaluation order. Within a subplan, the optimizer can analyze the resulting subplan complexity for each replica replacement (based on the formula presented in Section 5.4), and choose the less-impacting change. Before starting to split a task, the optimizer retrieves the unsolved replicated nodes and copy their cached plans into the task accordingly. If a cached plan is already evaluated, the optimizer just need to replace the replicated node by a **fetch** operator.

The second special occurence is of collateral annotations. The optimizer deals with this occurrence according to the size of the collateral plan. Small collateral plans can be attached to the main plan, thus enabling the optimizer to consider collateral annotations to determine an efficient invocation schedule. When splitting a task, to assign node heights, the root of an attached plan is handled as a child node. If collateral plans are potentially large (more specifically, when the node has $ch \geq k$), then the optimizer has to handle them independently. This is because we assume the entire collateral plan has to be executed *after* the node that triggers it. Since optimization and execution are interleaved

in our strategy, the optimizer can evaluate large collateral plans (including determining their split points) only immediately after their origin nodes. In a more relaxed execution environment, we can admit the optimizer processing both a plan operator and the dependencies of its collateral call concurrently. In such a scenario, all the collateral plans have to be attached to the main plan. However, we limit plan attachment to small collateral calls.

Figure 21 shows the split points of the task rooted by operator μ(sc5), from the initial plan in Figure 17. In this example, we also include the collateral plan rooted by μ(sc1.2). Nonetheless, we remark that μ(sc1.2) is not attached to μ(sc5). For simplicity, nodes are annotated with *ch* only if they have a collateral annotation, and with *lah* only if $lah \neq h$. Split points are indicated by large red arrows. We assume μ(sc5) is the first task to be evaluated, thus the operator ρ(sc8.1) is replaced by its corresponding cached plan. Moreover, we have that $k = 2$, and the split points are: μ(sc5), μ(sc7), μ(sc1.2), and μ(sc3.2). Notice that, if collateral dependencies can be evaluated concurrently, then the split points of the task of μ(sc5) are: μ(sc5), μ(sc7), and μ(sc3.2). Although these points do not seem to change much in this case, the optimization of their subplans is quite different. This is mainly because the optimizer will consider materialization alternatives for both μ(sc5) and μ(sc1.2) before executing them. Also, the descendants of μ(sc1.2) may be executed before μ(sc5).

Scheduling subplans evaluation. Basically, the subplans of a materialization task must be evaluated in topological order from the leaf nodes, such that child nodes are inspected first and collateral calls are properly triggered. This task traversal usually can be easily followed. However, since an abstract task does not enforce a total order on sibling operators, some subplans can be processed concurrently. Therefore, the optimizer has to determine an invocation schedule ($\succ$) for the task. This schedule does not need to be completely specified, since only split points have to be considered at this moment.

To focus the scheduling analysis on split points, the optimizer summarizes a materialization task with a *subplans guide* (or s-guide, for short), which is an access structure that expresses the dependency relationships between subplans. An s-guide contains only the split points of a task, along with their descendant relationships. Notice that two split points may be siblings in the s-guide even if their respective operators do not have this relationship in the materialization task. We consider two basic approaches to schedule the nodes of an s-guide. The first approach consists in determining an evaluation order on the children of each node of the s-guide. Such an order can be defined by some size-based heuristic, such as "*deepest first*" (*e.g.*, based on the absolute height of subplan root in the task), similarly to the size-based criteria used for tasks priority assignment in

```
1  function SplitTask(t, k): SplitPoints
2  {Returns the set of k-depth split points of task t.}
3  begin
4    let SplitPoints = ∅
5    {Handle replicated nodes}
6    for each cluster c in Clusters_t do
7      let Rep be the set of replicated nodes of c in t
8      if the cached plan of c is evaluated then
9        for each node v in Rep do
10         Replace the operator of v by fetch
11       end for
12     else
13       Choose a node v_m in Rep as the master replica
14       Replace v_m by the cached plan of c
15       for each node v in Rep do
16         if v ≠ v_m then
17           Replace the operator of v by fetch
18         end if
19       end for
20     end if
21   {Handle collateral annotations}
22   let Colls be the set of nodes containing collateral
         annotations in t
23   for each node v in Colls do
24     if ch_v < k then
25       let t_coll be the collateral plan of v
26       Attach t_coll to v
27     end if
28   end for
30   for each node v in t do   {search for split points}
31     let remainder = 0
32     if v has an attached plan then
33       remainder = mod(lah_v, k)
34     else remainder = mod(h_v, k)
35     end if
36     if remainder = 0 then
37       Add v to SplitPoints
38     end if
39   end for
40   let v_root be the root of t
41   if v_root ∉ SplitPoints then
42     Add v_root to SplitPoints
43   end if
44   return SplitPoints
45 end
```

Figure 20. Algorithm to compute split points.

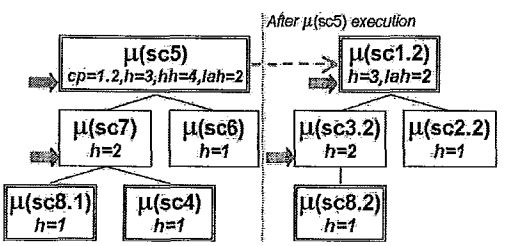


Figure 21. Split points of the materialization task rooted by μ(sc5), assuming that $k = 2$.

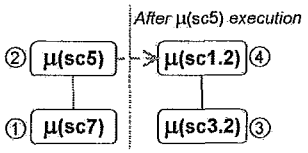


Figure 22. S-guide with evaluation order.

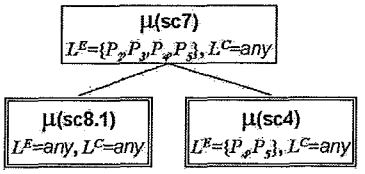


Figure 23. Abstract subplan rooted by μ(sc7) annotated with location scope.

Section 6.3. The second scheduling approach is based on a "*ready list*", which contains the subplans that can be immediately evaluated. Initially, the ready list contains all the leaf nodes of the s-guide. Once a subplan is evaluated, its parent node in the s-guide is added to the ready list. The optimizer makes scheduling decisions only for the subplans on this list; to choose the next subplan to be evaluated, the optimizer can apply some size-based heuristic on these subplans. Although this approach relies on localized decisions, since it ignores the overall task makespan, it can improve intra-task parallelism and workload distribution.

Figure 22 shows the s-guide for the task rooted by μ(sc5) of our running example, along with the collateral plan of μ(sc5). Circled numbers indicate the subplans evaluation order; nonetheless, only the topological order can be observed in this example, since this task does not have concurrent subplans.

Locating Web services. In AXML documents, Web services requests may be specified with abstract references, which need to be converted into some concrete service endpoints in order to generate physical materialization plans. The location scope of a plan is an essential input of our optimization problem. It determines a two-dimensional search space, since: (*i*) abstract references may be converted into many alternative addresses of Web service providers; and (*ii*) peers can collaborate to materialize an AXML document. Basically, the optimizer can retrieve this information from an internal peer catalog, a catalog server in the network (such as a UDDI server [52]), and from other peers. The optimizer tries to annotate operators with their respective execution and delegation scopes, primarily accessing the peer catalog and registered catalog servers. If some operators miss this information, then the optimizer can explore a dynamic discovery method, using the **locate** operator to gather the missing scopes from neighbors. This enables the optimizer to perform basic *contingency planning*. That is, the optimizer can try to automatically recover from failing in determining the providers of some Web services, by collaborating with other peers.

Since P2P systems are highly dynamic, the location scope of a materialization plan should be preferably provided by *late binding*. In our optimization strategy, this is done incrementally, for each subplan of a task, in some arbitrary tree walk. By restricting the location scope to subplans, we enable the optimizer to defer retrieving Web services addresses until they are really necessary. Hence, the optimizer can use more fresh (and reliable!) information.

When annotating the operators of a subplan with location scope, the optimizer replaces every μ operator that misses this information by a **locate** operator. After visiting the entire subplan, it checks if **locate** operators can be grouped by subtrees. This analysis aims at reducing communication costs. Moreover, the optimizer may decide to start evaluating another subplan (or other operators of the current subplan) while it waits for the result of a **locate** operator. Also, if too many operators of a subplan miss the location scope, it is quite likely that the optimizer will face difficulties evaluating the subplan, due to the lack of supportive information (*e.g.*, cost parameters and statistics). To solve this problem, either peers may embed supportive information in **locate** results, or the optimizer may delegate the entire subplan to another peer. In Figure 23, we show the subplan rooted by μ(sc7) annotated with the location scope according to the services distribution of Figure 1(b); we assume any peer can invoke all the requested Web services.

Generating and ranking physical subplans. Having specified the location scope of an abstract subplan, the optimizer is able to enumerate and analyze the costs of its physical alternatives. This is a key phase of our optimization strategy, and it mainly concerns reasoning about resource planning and invocation scheduling.

Algorithms for job scheduling usually rely on a *scheduling list* [33], which is essentially a sequence of nodes ordered by priority. To define the schedule, the first node is removed from the list and allocated to some available resource, repeatedly until the list is empty. The scheduling list can be either *static* or *dynamic*, according to whether the optimizer recomputes the priorities of unscheduled nodes after each allocation. Also, there are several different policies to assign priorities to nodes, most of them focused on the critical path of a plan. Notice that defining an invocation schedule depends on the available resources. Therefore, we consider the optimizer first chooses the peers that are going to participate in the materialization process, and then attacks the problem of finding a good invocation sequence based on the critical path.

Nevertheless, resource planning is also affected by the

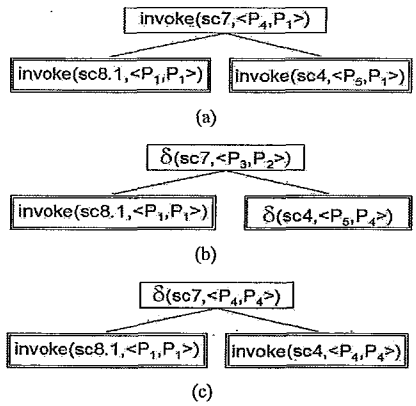


Figure 24. Some alternative physical conversions for the subplan rooted by μ(sc7).

```
1  procedure EagerEnumeration(sp,X)
2  {Enumerates the search space of the abstract subplan
     sp by applying a two-level cost analysis on its top
     X partial plans.}
3  begin
4    let Phys = ∅  {set of physical subplans}
5    Generate all the partial subplans of sp
6    Rank partial subplans by cost of potential transfers
7    let PP be the set of top X partial subplans of sp
8    for each subplan p in PP do
9      Generate physical subplans of p into Phys
10   end for
11   Rank subplans of Phys by makespan
12   Let M_best be the best subplan in Phys
13   Generate an invocation schedule for M_best
14 end
```

Figure 25. Eager enumeration of alternative physical plans.

order in which physical operators are processed. For example, if two concurrent operators are assigned to be executed at the same peer, they are probably going to be processed sequentially. Even if the peer offers the best execution cost for each operator, the overall performance may be worst than assigning one of the operators to another peer (and possibly profiting from parallel execution). To tackle this problem without having to previously define an invocation schedule, we rely on a cost model that considers the workload of operators assigned to peers (see Section 7 for further details). In general, the optimizer performs the following steps:

1. Generate the search space of *partial subplans*, whose nodes are set only with the execution scope;
2. For each partial subplan, generate the search space of physical subplans by setting the delegation scope of the algebraic operators; and
3. For each physical subplan, generate the search space of alternative invocation schedules.

However, we are considering decentralized execution environments, where an invocation schedule cannot be globally enforced. Hence, we focus our strategy on resource planning, assuming the optimizer chooses the "best" physical subplan and then determines a good invocation schedule only for this subplan. Observe that, by starting with partial plans based only on service providers, the optimizer is able to make use of some heuristics (*e.g.*, the "Context" heuristic [44]) to prune the search space of physical subplans, as in the eager enumeration approach presented next.

An abstract subplan works as a template for generating alternative physical subplans. A straightforward (and mostly inefficient) approach to produce these subplans consists in exhaustively enumerating the search space, such that the optimizer yields all the possible combinations of location scope and invocation schedule. To generate a physical subplan, first the optimizer converts every μ operator into an **invoke** operator by picking a definite invocation plan for the corresponding service call node from its $\widehat{IP}$. Once the location scope of the subplan is determined, the optimizer applies a transformation rule that searches for *delegation points*, namely the **invoke** operators whose caller is neither the master peer nor the caller of the parent node. This rule replaces **invoke** operators by δ accordingly. Figure 24 shows some alternative physical convertions for the subplan rooted by μ(sc7). The optimizer uses a cost model to rank these alternative physical subplans by their makespan.

Although exhaustive algorithms are often unfeasible for complete materialization plans, they may become useful in our optimization strategy, since the optimizer deals with subplans of reduced size. Furthermore, with our dynamic approach, the optimizer can use intermediary results to reduce error propagation in cost prediction, thereby improving the cost analysis of plan operators.

Another approach is based on an *eager enumeration* of the search space, where the optimizer generates alternative physical subplans only for the top X partial plans, as shown in Figure 25. The optimizer relies on a two-levels cost model. In the first level, only the execution scope of the operators is considered, and the optimizer tries to reduce the costs of transfering invocation results to the master peer, based on the set of distinct peers of each partial subplan. This selects subplans involving a few peers, which have a fast link to the master peer. Although clearly suboptimal, this heuristic may be efficient when communication costs are predominant. Notice that other criteria could be used to filter partial plans. In the second level, only the top X partial subplans are used to generate physical subplans, which are fully analyzed and ranked by their makespan. An eager approach is interesting when the size of the search space is critical even for low values of k.

It is worth mentioning that many different algorithms can be exploited to produce the search space of physical subplans. Yet, our work rather puts emphasis on breaking the optimization problem to reduce the search space, based on the structure of materialization plans. Still, most of these algorithms can perform efficiently with our strategy, since it enables them to handle smaller problems.

Evaluating physical subplans. An important feature of our strategy is that planning and execution are interleaved during the AXML materialization process. After selecting a physical subplan, the optimizer evaluates its operators in a bottom-up traversal, following the specified invocation schedule, as described in the algorithm of Figure 26. Recall that we consider a decentralized execution model where parts of a plan may be delegated to other peers. Therefore, the optimizer actually evaluates only:

- *local operators*, namely those whose caller is the master peer; and
- *delegation points*, which are represented by either δ or Θ operators.

Moreover, although delegated points may be nested in a subplan (*e.g.*, the δ operators in Figure 24(b)), we assume the optimizer does not reason about transitive delegation, and all of these points are supposed to return their result to the master peer. Nonetheless, it is worth mentioning that this restriction does not prevent peers from deciding to delegate parts of a subplan coming from another peer.

Evaluating a plan operator involves the steps shown in Figure 27. Basically, the optimizer builds the required inputs, invokes the corresponding service call, gathers the result, and updates both the materialization plan and the AXML document accordingly. Also, the optimizer has to check on the result to verify whether or not it contains intensional answers. If it is the case, the optimizer must update the subplan with the new service call nodes. To simplify the plan update, the optimizer can employ **pipe** operators based on the techniques presented in Section 4.4 for pipelined graphs. Namely, the optimizer generates an initial abstract plan for the intensional answer, and inserts each task found in the answer into the current subplan by connecting the new task roots as children of a **pipe** operator.

Notice that an intensional answer may significantly change the subplan, specially its height and consequently its complexity. Hence, the optimizer may have to review the subplan splitting to accommodate the new operators. Moreover, some optimization decisions may become wrong in the presence of new plan operators, and the optimizer may have to reconsider the chosen location scope and invocation schedule. Yet, in our strategy such an analysis does not ripple to the whole materialization plan. When an operator evaluation results in some intensional answer, the changes affect only the subplan that is being evaluated since the rest

```
1  procedure EvaluatePhysicalSubplan(v_root)
2  {Evaluates subplan rooted by v_root operator.}
3    begin
4    let Children be the set of child nodes of v_root
5    for each node v_child in Children in order of ≻ do
6      EvaluatePhysicalSubplan(v_child)
7    end for
8    if P^C_{v_root} = P_m or O(v_root) ∈ {δ, Θ} then
9      Evaluate v_root
10   end if
11 end
```

Figure 26. Algorithm to evaluate subplans.

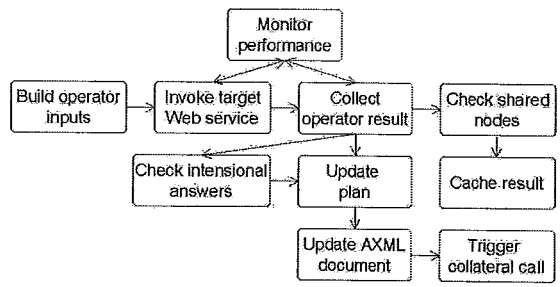


Figure 27. Steps of evaluating an operator of a physical subplan.

of the plan consists of abstract operators only. This avoids unnecessary re-optimization.

If the result of an operator evaluation corresponds to some shared node, then it has to be properly loaded into the cache. To manage keeping these results in the cache, the optimizer maintains for each cached plan a counter of *hanging references*, which represent the non-evaluated ρ and **fetch** operators of the main plan. Recall these operators stand for the replicated nodes of the plan. Initially, each counter is set with the number of respective replicated nodes. Whenever a replicated node is evaluated, its corresponding counter is decreased. The result of a shared node is kept in the cache while its counter is not zeroed. Some additional checking may also be applied to an operator result; for example, the optimizer may want to validate the result against some expected datatypes before updating the materialization plan and AXML document.

During an operator evaluation, the optimizer monitors the performance of both service invocation and result transfer. It may ask for subplan re-optimization in case the performance significantly surpasses the expected costs. Alternatively, the optimizer may exploit some rescheduling technique, similarly to query scrambling [53], to hide evaluation delays. Furthermore, if some error situation arises, the optimizer may resubmit an operator for evaluation (considering the user allows it). For example, an operator re-evaluation may occur due to service execution error, communication

timeout, or insufficient/incorrect result (w.r.t. some predefined criteria, such as expected datatypes).

Once an operator is successfully evaluated, the optimizer must check if it has a collateral call to be triggered. Nonetheless, this is done only for local operators, since we assume that delegating subplans includes processing their collateral calls remotely.

Foreseeing intensional answers. An advantage of our optimization strategy is that intensional answers affect only the current subplan evaluation. However, we consider that the optimizer reacts to intensional answers just when they occur. Namely, it does not try to foresee whether a Web service may return AXML data, using this information for performance prediction. An alternative for the optimizer to encompass intensional answers in the physical subplans enumeration is to look at the expected result type of the required Web services, as proposed in [3].

Our strategy can be extended to support this feature by exploiting the idea of enabling condition behind collateral calls. That is, likewise a collateral call, an intensional answer represents service call nodes that have to be invoked after their origin call. Thus, the optimizer may insert some "special collateral calls" into the materialization plan to represent intensional answers. An additional enabling condition must be specified for these special calls, to guarantee that they are going to be triggered only if they are actually returned as the evaluation result of their origin node. Notice this may significantly inflate in advance the size of a materialization plan. Although our dynamic optimization strategy is quite adequate for large plans, handling intensional answers *a priori* is rather a complex subject, and we leave a deeper analysis of this issue as future work.

6.5 Delegating AXML Optimization

The algebra proposed in Section 6.2 contains some operators specially tailored for P2P collaboration. They represent different collaboration possibilities to process service calls of an AXML document, namely of delegating: plan optimization (Θ); plan evaluation (δ); and Web service location (**locate**). These collaboration operators are executed remotely by their target peers, and their results are properly merged into the materialization plan by the master peer. Hrom the perspective of the master peer (P_m), evaluating these operators involves three basic phases:

1. *Target selection*, that is when P_m identifies possible collaborators, and selects a target peer among them to handle the subplan rooted by the delegated operator;
2. *Collaboration contracting*, when P_m builds the delegated subplan (including all the necessary input data), and sends it to the chosen target peer; and
3. *Result delivery*, when the target peer sends the result of the delegated subplan back to P_m.

Next we discuss issues involved in each of these phases.

Selecting target peers. This phase focuses on providing an execution scope for the collaboration operator. Observe that both the execution and the delegation scope of a collaboration operator may differ from those of its service call node. By default, the caller of a collaboration operator is always the master peer of its subplan, since we consider the master peer cannot enforce transitive delegation (*i.e.*, the delegation scope is empty). That is, we assume:

$$L^C_{op} = \{P_m\}\ ,\ \ if\ \ op \in \{\delta, \Theta, \textbf{locate}\}. \qquad (11)$$

Here we denoted the delegation scope L^C_{op} of an operator op similarly to the notation used for service call nodes, as defined in Section 5.

Going further, collaboration operators request P2P-specific Web services, which are provided by AXML-enabled peers. Therefore, their execution scope is mostly determined by the peers in $\mathcal{N}$. A particular case is that of δ operators, which are used mostly to reduce data transfer costs. The executor of a δ operator is chosen essencially from the delegation scope of its service call node, assuming the corresponding peers support collaboration. Namely, given $\delta(v, IP_v)$, we have that:

$$P^C_\delta = P_m \quad and \quad P^E_\delta = P^C_v \quad , \qquad (12)$$

where $P^C_v \in (L^C_v \cap \mathcal{N})$.

For Θ and **locate** operators, there is seldom a direct relationship between the execution scope of the collaboration operator and its service call node. Thus, the execution scope is usually arbitrarily chosen by the master peer based on some QoS metrics. For instance, the optimizer may use SLA (Service-Level Agreement) specifications of the P2P-collaboration Web services to prune target candidates, similarly to the approach proposed in [35]. Target selection can be either: *static*, if it is based on existing statistics and costs; or *dynamic*, when the optimizer polls specialized servers on the network for fresh information on target candidates. Dynamic selection may become necessary in scenarios such as ad-hoc P2P systems. Nevertheless, it is usually quite expensive due to the communication costs of its required control flows, and it must be carefully explored (*i.e.*, mostly in case of missing or highly-outdated statistics). Although our optimization strategy can support both static and dynamic target selection, for simplicity we considered only the static approach. Namely, we assume P_m always chooses targets from $\mathcal{N}$.

Since a materialization subplan may contain several collaboration operators, the optimizer can decide to set target peers either for the entire subplan or just before evaluating each operator. Furthermore, target selection may require some P2P negotiation to ensure the chosen peer is available (and willing!) to receive and process the delegated subplan. This additional step consists basically in contacting the target peer to confirm its participation.

Collaboration contracting. Once a target peer is chosen, the master peer has to properly build the delegated subplan. That is, the optimizer has to produce a subplan containing:

- the collaboration operator, along with all its (active) descendant nodes in the materialization plan;
- the data elements that should be passed as input of service call nodes. To retrieve these elements, the optimizer may have to evaluate XPath expressions for non-concrete parameters; and
- the collateral calls, along with their dependencies (if any).

We assume all local nodes are evaluated before sending a delegated subplan. That is, only their results are embedded into the subplan. Furthermore, since usually there is a mismatch between a materialization plan and the AXML tree of its service call nodes, a delegated subplan follows the plan structure shown in Figure 17. Such a structure organizes the plan into three areas, and it aims at avoiding unnecessary node replication. This is very important to reduce communication costs.

Also, some sideways information can be passed embedded into a delegated subplan. For example, to avoid delegated subplans to be endlessly forwarded, we assume a *hops counter* goes along on each subplan. In this case, a hop represents each time a subplan (or a part of it) is delegated to another peer. This information is related to the *delegation trace* of a subplan, which indicates all the peers that have ever processed it.

Before sending a delegation subplan to its target peer, the optimizer has to serialize it (in the AXML format) and marshal the resulting AXML data into a SOAP envelope. A serialized subplan contains both a header and a body section. The subplan header keeps general properties of the subplan, such as its delegation trace and some optimization hints (*e.g.*, previously estimated plan costs), whereas the body encodes the plan operators and their input data. On the target peer side, a subplan has to be unmarshaled and parsed before resuming the materialization process. Notice these operations incur processing costs which the optimizer has to consider when analyzing materialization alternatives, as described in Section 7.

Result Delivery. When the master peer receives the result of a delegated subplan, it has to parse the serialized SOAP response, insert the embedded service results into the AXML document (if necessary), and update the materialization plan accordingly. The materialized contents that are embedded into the result of a subplan consists of a forest of nodes. The master peer uses a "origin" parameter to identify the respective service calls in the AXML document.

While the **locate** operator aims at retrieving only supportive information for the delegated subplan, both δ and Θ operators may involve some AXML materialization. In case the result contains materialized contents, it is basically composed by the results of:

- the children nodes of the collaboration operator;
- the operators related to persistent service call nodes; and
- all the collateral calls triggered in the subplan.

Recall that persistent nodes are essentially first-level service calls and shared dependencies. Also, notice that although temporary results are not required in the result of a delegated subplan, we consider the results of their collateral calls are sent back to the master peer.

Horizontal plan partitioning. The Split algorithm breaks materialization tasks in depth, to attack the optimization problem incrementally. However, if some plan operator has too many children, such a technique is not effective. To overcome this drawback, the optimizer may also partition a materialization subplan horizontally by inserting some collaboration operators. In particular, the Θ operator enables the master peer to descentralize the optimization of a plan.

Inserting Θ operators in a subplan is similar to procedure used to update a dependency graph with *pipe* nodes. The operators that are going to be remotely evaluated are connected as children of the Θ operator.

To decide when horizontally partition a subplan, the optimizer can either use the resulting complexity of the subplan or some fixed horizontal splitting parameter k_h. Futhermore, instead of asking a target peer to return only one physical plan as the result of a Θ operator, the master peer may consider getting a set of alternative solutions. Peers may also agree in setting a limit for the maximum number of inspected alternatives for a delegated subplan, thus limiting the expected optimization time.

7 Cost Analysis of AXML Materialization

Although heuristics are very usefull when dealing with complex optimization problems, in order to compare the performance of alternative materialization subplans, the optimizer requires objective metrics. In this Section, we present a set cost formula to model the performance of AXML materialization. The proposed cost model considers relevant aspects, such as:

- heterogeneous machines and communication links;
- equivalent Web services;
- subplans delegation;
- parallel execution;
- invocation dependencies and collateral calls; and
- processing workload of peers.

As we discussed in Section 5.4, the performance of alternative physical plans is represented by their makespan. This metric represents the clock-time spent from the start of the materialization process to the moment when the master peer gathers into the AXML document the results of all its embedded service call nodes. Intuitively, the makespan is determined by accounting the costs of Web services invocations/executions, and of communication messages between peers. To estimate the makespan of an entire plan, these costs must be properly combined according to the plan operators and their inter-relationships.

On the other hand, the optimization strategy of XCraft is based on dynamic and incremental plan generation. The optimizer has to analyze plans that contain abstract operators. To cope with this incremental approach, in XCraft we adopt a *multi-leveled cost analysis* based on three distinct cost models. In the first level, the optimizer estimates the costs of a materialization plan in terms of its complexity (*i.e.*, the size of its search space). For this purpose, we use the complexity bounds determined in Section 5.4. This enables the optimizer to avoid processing plans with too expensive analysis.

With the second-level cost model, the optimizer can limit the plans analysis by using some heuristic criteria. In particular, we assume it estimates costs for partial plans. In this case, the optimizer considers only the execution scope of plan operators. The idea is to emphasize the proximity (in terms of communication costs) of peers that are candidates to execute the requested Web services, with respect to the master peer. This way, the optimizer can rank partial plans before generating their physical alternatives. Finally, the third-level cost model consists of a comprehensive response-time analysis of plan operators.

We describe in Section 7.1 basic cost ingredients. The overall formula used in the second-level cost analysis is presented in Section 7.2, while Section 7.3 details the main components of the third-level cost model.

7.1 Representing Heterogenous Scenarios

Traditional cost models usually take into account very detailed information on the machines, such as I/O and CPU operations costs. Since P2P systems are quite heterogeneous and with autonomous peers, gathering this information is seldom possible. In [44], we modeled the basic costs of invoking a Web service by focusing on the response time of its major operations. According to [44], costs are computed from the client viewpoint. The response time of a service call v is denoted by $ccost(v, P_i, P_j)$, where P_i is client peer and P_j is the executor of v.

We consider that costs are given in time units and that peers may have different performance capabilities (CPU clock, RAM memory, etc.). For simplicity, we assume they are sequential machines, namely they can execute only one service call at a time and requested service executions have to join a queue at each peer. The *execution queue* of a peer P is an ordered list of service executions denoted by q_P. We assume execution queues are infinite and work on a "first arrived, first served" basis. The term $|q_P|$ represents the lenght of q_P, that is the number of service executions waiting in q_P.

The proposed cost model is sensitive only to *inter-operator parallelism*, such that a single service execution cannot be split among peers. Also, service executions are *non-preemptive*, namely they cannot be interrupted to be resumed by another peer. We denote by $ET(v, P)$ the average execution time of the service call node v at peer P. The terms $callSize(v)$ and $resSize(v)$ represent the size in bytes of the input and output parameters of v, respectively.

We expect peers interconnected through heterogeneous links. Therefore, the costs of transfering X bytes of data from peer P_i to P_j costs is:

$$net(X, P_i, P_j) = \frac{X}{B(P_i, P_j)} \quad , \qquad (13)$$

where $B(P_i, P_j)$ denotes the bandwidth of the link from P_i to P_j. Notice that $B(P_i, P_j)$ may be diffent from $B(P_j, P_i)$. It is worth mentioning that both the input parameters and the result of a service call may involve large data transfers. For example, in Figure 2(a), the input parameter of the call sc9 is expected to be a PDF file with possibly a few Mega bytes, though its result is only an excerpt of the input.

7.2 Heuristic Cost Analysis

This cost analysis points out materialization plans involving fewer peers, as well as peers that are close (in terms of communication costs) to the master peer. It is worth noting this heuristic selection tends to stretch out the makespace of the resulting physical plans. This happens because communication costs are more weighted, and parallel executions may be ignored. Nonetheless, in very large search spaces or in scenarios with high communication costs, this preliminary cost analysis can be helpfull.

Let be the set of distinct peers involved in a materialization plan $\mathcal{M}$. The heuristic cost of $\mathcal{M}$ is:

$$hcost(\mathcal{M}) = \sum^{\forall P_i \in DP_{\mathcal{M}}} net(ARS, P_m, P_i) \quad , \qquad (14)$$

where:

- $DP_{\mathcal{M}}$ is the set of distinct peers in the execution scope of $\mathcal{M}$;
- P_m is the master peer of $\mathcal{M}$; and
- ARS is a constant for the average result size of service calls in $\mathcal{M}$.

Notice that usually the heuristic costs of an entire plan can be quickly estimated.

This heuristic cost model is used to rank plans previously to a detailed analysis. For instance, the optimizer may apply this analysis first, and then calculate detailed costs only for n best alternative plans.

7.3 Costs of Plan Operators

Given a materialization plan $\mathcal{M}$, the cost of an operator $op \in \mathcal{M}$ is estimated as:

$$cost(op) = ccost(op) + dcost(op) + \sum^{\forall op_{cc},\, op \hookrightarrow op_{cc}} cost(op_{cc}) \quad , \quad (15)$$

where:

- the term $ccost(op)$ is the client-side cost, as provided in [44];
- the term $dcost(op)$ is the cost of the dependencies of op; and
- $op_c c$ indicates collateral calls.

Remember plan operators correspond to service call invocations, thus the cost analysis is uniform for different operator types. Therefore, the overall cost of a materialization plan $\mathcal{M}$ can be calculated recursively as:

$$overall\_cost(\mathcal{M}) = \sum^{\forall r_i \in ROOT_{\mathcal{M}}} cost(r_i) \quad , \quad (16)$$

where $ROOT_{\mathcal{M}}$ is the set of root operators of $\mathcal{M}$.

Invocation dependencies. Since peers have processing queues, if two service executions are mapped the same peer, they run sequentially. Thus, to properly estimate the costs of invocation dependencies, it is necessary to consider the processing workload of peers. We restrict this analysis to the children of each operator in a materialization plan.

Given a plan operator op, we take the set DP'_{op} of distinct peers involved in the evaluation of its dependencies. Notice DP'_{op} does not concern the location scope of op. Then, for each peer P_i in DP'_{op}, we calculate its total processing load as:

$$total\_load(P_i) = \sum^{\forall op_j \in Children(op, P_i)} cost(op_j) \quad , \quad (17)$$

where $Children(op, P_i)$ is the set of invocation dependencies of op that are executed by P_i. From this result, we can estimate the cost of the dependencies of the operator op as:

$$dcost(op) = \max^{\forall P_i \in DP'_{op}} (total\_load(P_i)) \quad . \quad (18)$$

It is worth noting that if operators run in blocking mode, then we have:

$$dcost(op) = \sum^{\forall op_j \in Children(op)} cost(op_j) \quad , \quad (19)$$

where $Children(op)$ is the set of all the dependencies of op. In this case, parallel executions are disregarded.

Peer energy factor. For P2P systems with low bandwith rates, the optimizer may tend to sacrifice parallel execution for the sake of avoiding data transfers. In this context, peers may be overloaded with service assignments. Thus, for fair costs ranking, the optimizer has to consider performance penalties from peers workload. When estimating the costs of a plan operator op, we define the energy factor of a peer P_i in DP'_{op} as:

$$ef(P_i) = \frac{Bogo_{P_i}}{\sum^{\forall op_j \text{ with } P_i \text{ in } IP_{op_j}} ET(op_j, P_i)} \quad , \quad (20)$$

where $Bogo_P$ is the BogoMips [58] speed of P_i. The energy factor can be used to add some extra response time for service executions at P_i, which are due to its performance penalties. Another (simpler and more imprecise) way to estimate this factor would be computing the inverse of the number of plan operators assigned to the peers.

Delegation costs. Delegating a subplan encompasses the costs of: (i) sending the plan along with its input data; (ii) evaluating the plan at the remote peer;(iii) returning the plan along with its persistent results back to the master; and (iv) updating the AXML document. This means the costs of δ operators can be computed by Equation 15, with small changes in the estimation of input and result sizes. In particular, the input size must take into account the results of dependencies that were previously evaluated.

8 XCraft Architecture

We present a service-oriented optimizer architecture called XCraft, which enables dynamic and decentralized materialization of AXML documents, and supports the proposed optimization strategy. XCraft works in a multi-thread fashion, as a facade component of the ActiveXML peer; it interacts with the AXML document repository, the services and statistics catalogs, and the Service Call Handler.

Main XCraft modules. Figure 28 shows the main modules of the XCraft optimizer, which conducts AXML materialization as follows. When the contents of an AXML document are requested, its master peer starts a new optimization task at XCraft. The *Graph Extractor* analyzes the service calls embedded into the document and produces its dependency graph (or retrieves it, if it is already available). This graph is used by the *Abstract Plan Builder* to extract the corresponding MFST and to yield an initial abstract plan. The *Planner*, a central XCraft module, takes the

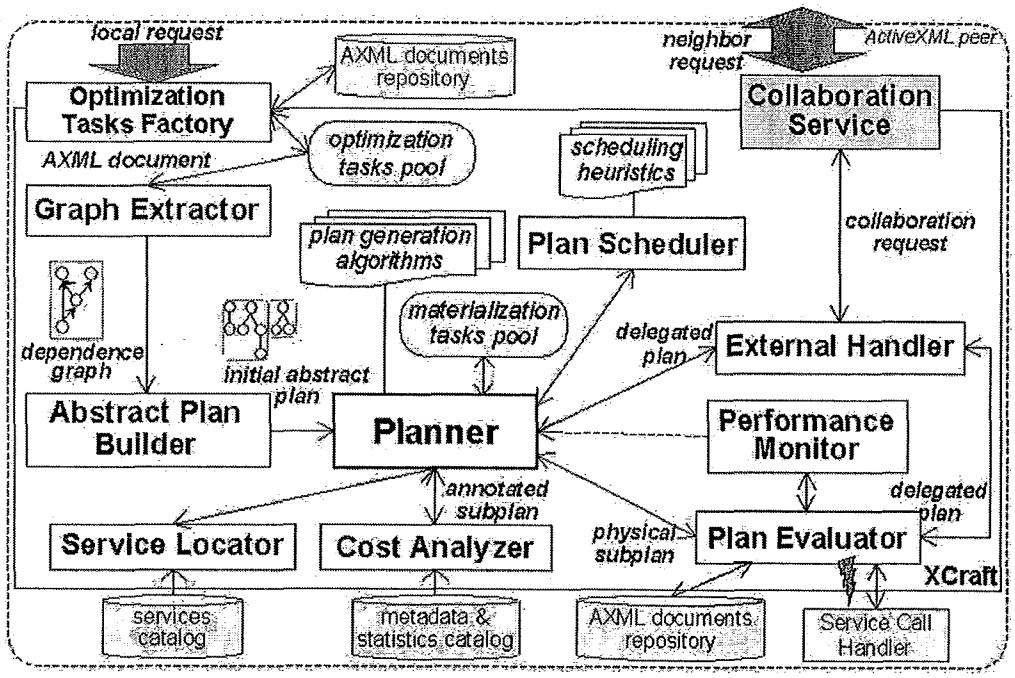


Figure 28. System architecture of the XCraft optimizer.

initial plan, breaks it into materialization tasks and calculates their priority.

Materialization tasks are processed such that each task is split into subplans, and each subplan is optimized and completely evaluated before optimization is resumed. First, the Planner asks the *Service Locator* to identify both the execution and the delegation scope of the current subplan. According to the plan generation strategy, the Planner roves the search space of alternative physical plans; it uses the *Plan Scheduler* to determine $\succ$ by applying some scheduling heuristic. The Planner asks the *Cost Analyzer* to estimate the makespan of physical subplans, registering the subplan with the best makespan during the search. Then, it sends the overall best subplan to the *Plan Evaluator*, which executes it and returns the results, possibly along with intensional answers.

As the evaluation proceeds, service call results are gathered and merged into the AXML document. Furthermore, the *Performance Monitor* watches over operators evaluation, and it can trigger subplan re-optimization if necessary. Both the Planner and the Plan Evaluator may also get plans coming from the *External Handler*, which processes requests from other peers. These requests are received by the *Collaboration Service*, which implements the interface of the basic Web services for P2P collaboration. For collaboration requests, evaluation results are sent back to the origin peer in the Collaboration Service reply.

For simplicity, we omitted in Figure 28 two modules of the XCraft architecture: the *Plan Cache*, where the optimizer keeps shared plans and their results; and the *Optimizer Profile Loader*, which sets relevant properties to configure the behavior of the XCraft internal modules (*e.g.*, the heuristic to be used by the Plan Scheduler).

Configurable optimizer profiles. There are many variables to be considered by the XCraft optimizer when materializing an AXML document, such as:

- the subplans depth used by the splitting algorithm;
- the scheduling heuristic;
- if re-optimization is allowed;
- if peer can forward delegated subplans to other participants; and
- the heuristic used to generate the search space of alternative physical subplans.

To handle all these options, XCraft uses the notion of *optimizer profile*, that is a set of properties that control the optimizer behavior. The profile is loaded at launch time, but it can be updated during the peer life cycle. It provides flexibility to adapt to different application requirements.

Basic optimization services. Cost parameters and statistics are provided by some basic optimization services, which are usually available at any ActiveXML peer. Although

```
<serviceDefinition type="query"
   axml:docName="UnionService"
   xmlns:axml="http://www.activexml.net/AXML">
    <parameters>
        <param name="_documentIn"/>
    </parameters>
    <definition>
      <query> <![CDATA[
        {select e1
         from e1 in SigmodRecord//articlesTuple}
           union
        {select e2
         from e2 in {_documentIn}//articlesTuple};
      ]]> </query>
    </definition>
</serviceDefinition>
```

Figure 29. Declarative Web service with union operation.

```
<?xml version="1.0" encoding="UTF-8"?>
<serviceDefinition type="query"
  axml:docName="JoinService"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:axml="http://www.activexml.net/AXML">
    <parameters>
        <param name="_documentIn"/>
    </parameters>
    <definition>
      <query> <![CDATA[
        select e1
        from e1 in SigmodRecord//articlesTuple,
          e2 in {_documentIn}//articlesTuple
        where e1/title/text() = e2/title/text();
      ]]> </query>
    </definition>
</serviceDefinition>
```

Figure 30. Declarative Web service with join operation.

the optimizer may collect some missing costs and statistics at runtime, much of the supportive information that is required during the optimization process must be gathered in advance. Furthermore, plan operators such as Θ and δ are implemented as basic Web services for collaboration, thus enabling peers to exchange evaluation plans encoded as AXML data.

9 Experimental Results

We have implemented and tested the proposed optimization strategy in the ActiveXML system [13]. We extended the ActiveXML peer (version 4-Beta) with the XCraft optimizer components presented in Section 8. We used the Java language and open-source software, such as Apache Tomcat 4.1.29, JDK 1.4.2, and Axis 1.1. To compute spanning trees, we used a Java implementation of the Prim-Jarnik algorithm [31].

In our tests, we deployed three ActiveXML peers extended with the XCraft optimizer and some basic collaboration Web services. At each peer, we also deployed two declarative Web services, which perform respectively a union and a join operation on documents derived from the ACM SIGMOD Record articles database [48]. The specifications of these declarative services are described in Figures 29 and 30. They take a single input parameter named "_documentIn", which is combined (either by a union or a join operation) with a locally stored file. Notice the "axml:docName" parameter (of the "serviceDefinition" element) indicates the name of the declarative service. In the ActiveXML plataform, the query of declarative Web services is written with the X-OQL language [64].

We used three heterogeneous machines under different workloads, as described in Figure 31. Processing power is represented by BogoMips [58]. *Master* indicates the master peer, which is connected through a 512Kbps Internet link to the other two machines. *Laptop*1 and *Laptop*2 are located in a 36Mbps local network. Figure 32 shows these peers connections.

Peer	O.S.	BogoMips	RAM
Master	Debian GNU/Linux	2957.31	512Mbytes
*Laptop*1	MS Windows™XP	1718.18	512Mbytes
*Laptop*2	Linux SuSe™	1198, 77	512Mbytes

Figure 31. Hardware of deployed ActiveXML peers.

We generated sets of AXML documents with different configurations of service call nodes by varying the height and width of the document trees. Recall the height is determined by invocation dependencies and the width corresponds to the number of trees in the MFST of the AXML document. Basically, in the experiments we used AXML documents that contain sequences (*i.e.*, batch pipelined tasks) and parallel splits patterns from grid workflows [54].

We performed two basic analysis. First, we identified aspects that have relevant impact on the materialization complexity (*i.e.*, the number of alternative plans). We observe the time spent in optimization with different plan generation approaches, and compare these results with the XCraft dynamic strategy. In the second battery of tests, we evaluate the gains achieved by subplan delegation. We focused on

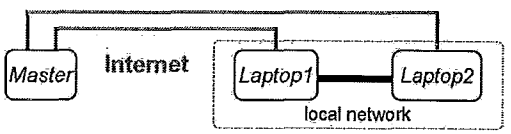


Figure 32. P2P network used in experiments.

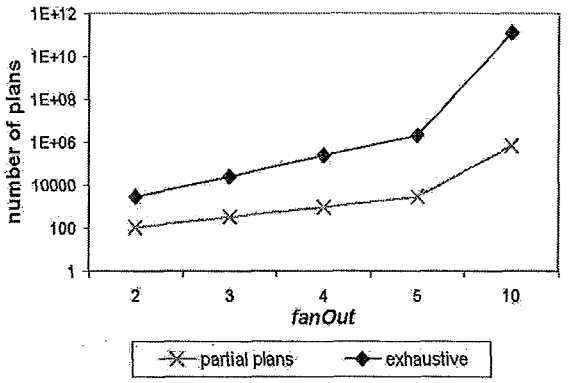


Figure 33. Impact on the size of the search space by varying the $fanOut$ of service calls.

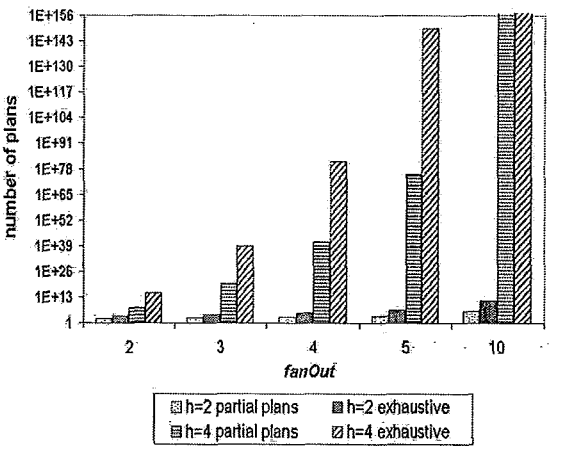


Figure 34. Search space analysis from varying both the document height and the $fanOut$.

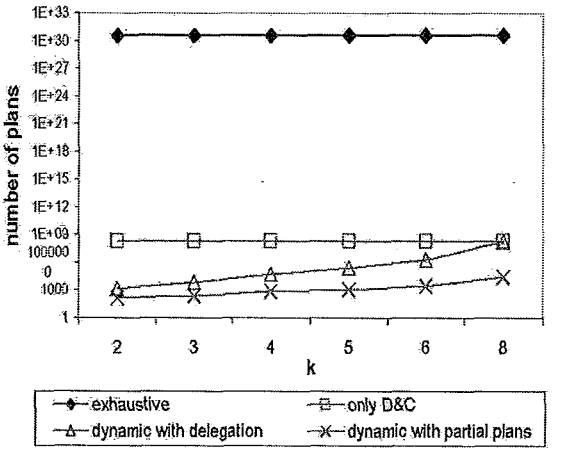


Figure 35. Search space reduction in XCraft.

delegation of service invocations, since both optimization and service location operators are more related to contingency planning and do not directly reflect performance improvement. These operators rather improve the adaptivity capabilities of the optimizer.

It is also worth noting that, in P2P and grid systems, the communication costs from transfering data between two nodes that are delegated to the same peer are usually assumed to be zero. Nonetheless, this simplification is not true for Web services, since they always involve heavy operations for XML handling, as observed in [44].

9.1 Devising the Search Space

The time spent on the optimization process is a crucial point when efficiently materializing AXML documents. Since this represents an NP-complete problem, exahustive search is usually prohibitive, making the use of heuristics mandatory. Moreover, in P2P systems, the optimization process itself cannot be time-consuming due to the dynamic behavior of peers. Therefore, an important task of the optimizer is to analyze the size of the search space of a given materialization plan.

The great improvements of hardware performance have made possible to tackle several complex optimization problems. Nonetheless, we observed they are still insufficient to solve the issues posed by AXML materialization, which usually involves very large search spaces. To have a more clear idea of the size of the search space of an AXML document, we used the complexity formulas presented in Section 5.4 to identify its relevant dimensions and estimate their impact. For instance, in Figure 33 we varied the $fanOut$ of service call nodes for an AXML document with height $h = 2$ and four first-level service calls (*i.e.*, width is $|\Lambda| = 4$), considering three peers. Notice the axis of number of plans is in a logarithmic scale. Results corresponds to the complexity of partial plans with only executors and to the complete search space, assuming an exhaustive method. Figure 34 shows the size of the search space by varying both the $fanOut$ and the document height. The search space grows exponentially with respect to both $fanOut$ and h. In fact, even for small documents, its size is significantly large. In further analysis, we found this exponential behavior stands the same for the number of peers involved in the materialization process.

XCraft uses a dynamic optimization strategy to reduce the number of inspected plans, yet taking into account relevant properties such as communication costs. Basically, XCraft breaks materialization plans according to a given parameter k. We can observe in Figure 35 the impact of our strategy on the number of inspected alternative plans. We used an AXML document with four first-level service

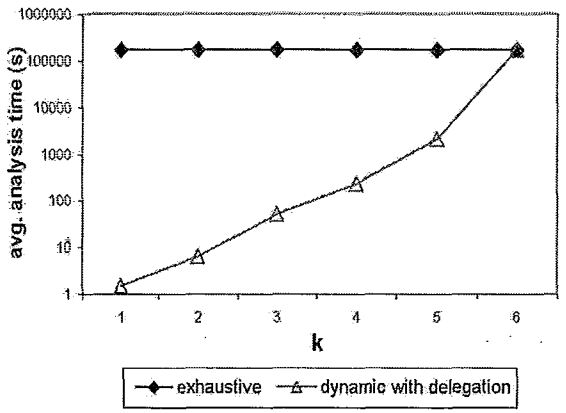


Figure 36. Simulated optimization time.

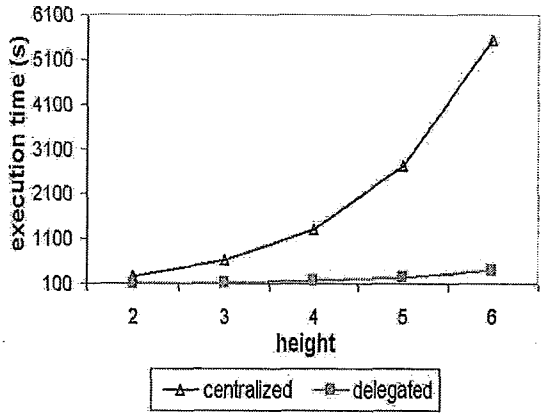


Figure 37. Performance gains obtained by subplans delegation.

calls of height fixed at $h = 8$. We assumed a very simple case, where the $fanOut$ of each service call node is 1 (namely, the document contains only 32 service call nodes). We estimated the number of inspected plans with our dynamic strategy for different values of the k parameter (*i.e.*, the height of each subplan to be analyzed by the optimizer), considering both the analysis of subplan delegation and choosing only service executors. We also compare these results with the exhaustive search strategy, and results of using the Divide&Conquer heuristic to identify independent materialization tasks. Observe that even in this simple case, the size of the search space prevents adopting the exhaustive strategy. Our dynamic approach provides XCraft with flexibility to deal with complex AXML scenarios, by allowing it to scan search spaces with manageable sizes.

The size of the search space has a major impact on the optimization time. We simulated this time by considering the optimizer spends an average of 0.5 millisecond to generate and analyze an alternative plan. We came to this value by observing experimental results obtained with small AXML documents. Although larger documents tend to require more time to be generated and analyzed, this average metric sets a good performance reference, as shown in Figure 36. We considered the same AXML document used in Figure 35.

9.2 Plan Delegation Effects

Although our dynamic strategy produces suboptimal solutions, it enables the optimizer to exploit subplans delegation, which usually results in significant performance gains. This can be noted in Figure 37. We evaluate the performance achieved by delegating materialization subplans containing service call nodes with $fanOut = 1$, and invocation results with 100Kbytes. This corresponds to AXML documents that contain batch-pipelined tasks (*i.e.*, with at most one dependency). We vary the height of nested service calls in the document from 2 to 6.

In a centralized evaluation strategy, the optimizer invokes of each service call, and gathers their results from an Internet link. With delegation, the master peer sends physical subplans (*i.e.*, materialization tasks) to be evaluated remotely, and receives only persistent service results to compose the final document contents. Only the root element of each materialization task is a persistent result. It is worth noting that, for higher values of $fanOut$ and of the size of service results, the performance gains of plan delegation tend to be even more expressive.

10 Related Work

Materializing AXML documents is quite similar to executing workflows: embedded service calls are tasks to be performed, which are often related to each other, causing some invocation constraints and data flows. These invocation constraints correspond to some basic control flow patterns, namely sequence, parallel split, and synchronization [54]. However, AXML materialization always involves some data flows towards the peer that is gathering the document contents (called *master peer*). Hence, an AXML document can be incrementally composed and consumed, while partial results are seldom meaningful in workflow systems.

We represent AXML invocation constraints in a formalism based on *directed acyclic graphs* (DAG), similarly to models used for business processes orchestration in workflow systems [15, 33, 54]. As in scheduling workflow tasks for grid computing [15, 38, 56], we are interested in determining an efficient assignment of tasks (Web service executions) to distributed resources (peers). However, in grid systems usually tasks are assigned to *sites* whose infras-

tructure encapsulates many servers, aiming mainly for load balance [15, 27, 42, 56]. Still, planning workflows in distributed heterogeneous systems is an NP-complete problem [33], which remains a research challenge. Likewise, optimizing AXML materialization is a hard problem, with additional complications from the volatility of a P2P scenario.

Allocating resources and scheduling tasks to efficiently execute workflows is indeed an important issue. Current planners [15, 27, 42, 56, 61] are essencially concerned with heuristics to schedule tasks and algorithms to improve locality of required data files. Nonetheless, tailored for grid computing, these planners are often based on static analysis [15, 27, 42, 61]. In AXML materialization, besides the performance and membership fluctuations of the system, the optimizer has to be prepared for occasional changes in the materialization plan due to intensional answers. On the other hand, planners that are based on dynamic strategies do either greedy [56] or opportunistic [38] resources selection. Since they work with local decisions, they usually cannot explore avoiding unnecessary data transfers. Even when planners are dynamic or adaptive [27, 38, 42, 56], they consider either centralized or hierarchical coordination, and rely on re-optimizations to react to changes.

Notice that, although decentralization has become a key feature in both P2P and grid computing, current systems do not support a decentralized planner. Our results highlight promising performance gains achieved by a decentralized approach.

AXML documents are similar to decision flows [30] in the sense that their materialization is *attribute-centric*, namely it aims at determining the values of certain data elements. Yet, conversely to [30], our strategy is dynamic and enables decentralized evaluation.

Previous work on AXML optimization mostly addressed typing control [39], XML query processing [3], and data and Web services replication [7]. Mechanisms to generate alternative strategies for AXML materialization, including basic cost formula for performance prediction, was first presented in [44]. XCraft is built upon these ideas, and focuses on the problem of efficiently producing and evaluating materialization plans in dynamic P2P systems. Recently, Abiteboul *et. al* [9] proposed an algebraic framework to generate AXML materialization alternatives, with emphasis on Web services that can be described by queries. In XCraft, we consider issues related to handling search complexity, resources heterogeneity and P2P membership dynamics when generating materialization plans.

11 Conclusions

Materializing an AXML document corresponds to a general case of finding an efficient assignment of inter-related tasks to heterogeneous machines, which is an extremely hard optimization problem. Nevertheless, this became a current challenge for many information integration systems based on Web services, such as P2P systems and grid computing. In this paper, we presented an optimization strategy for AXML materialization, which widely explores dynamic techniques, thus scaling well for decentralized and ad-hoc systems. We believe this work goes beyond the context of AXML documents, and contributes to the efficient instantiation of abstract workflows, specially in highly-dynamic and heterogeneous systems. Also, with a descentralized architecture for collaborative optimization, we highlighted an important issue that has been neglected in most of the current systems.

In XCraft, since we assumed the cost analysis of materialization subplans is exhaustive, the algorithm used to generate these subplans is quite sensitive to the choice of the height of the planning step (*i.e.*, the k parameter). To diminish this shortcoming, we are currently developing methods based on stochastic algorithms and local search, such as the techniques proposed in [63]. The overall idea is to incrementally refine an arbitrary initial solution until some condition is satisfied (*e.g.*, some bounded period of time or performance improvement percentage), while allowing the optimizer to ramdomly move in the search space based on some probability function for plan acceptance. As stated in [45], these metaheuristics are usually very suited for searching good solutions using non-monotonic cost functions (as in AXML materialization).

There are many interesting paths to pursue the ideas raised in this paper. We are considering to extend the optimization strategy to support contingency planning for service call failures, that is to generate branching plans taking some or all of the possible alternative evaluations into account, as presented in [23] for Web services composition. Also, materialization plans are a very graphical and intuitive representation that can be explored to monitor AXML materialization, possibly allowing the user to interfere in the process "on the fly". Finally, we have observed that planning and scheduling Web service invocations are quite affected by resources availability. Hence, an interesting research perspective consists in investigating non-intrusive techniques for resources provisioning in P2P systems.

Acknowledgements. The authors thank CNPq agency for partially funding this work. Gabriela Ruberg is also supported by the Central Bank of Brazil. The contents of this document express the viewpoint of the authors, and do not represent the position of either these institutions. We specially thank Ioana Manolescu and Serge Abiteboul for fruitful discussions (and text revisions) on preliminary versions of this work. Finally, we thank the Gemo team, at INRIA-Futurs, for the ActiveXML open-source prototype.

References

[1] S. Abiteboul, B. Alexe, O. Benjelloun, B. Cautis, I. Fundulaki, T. Milo, and A. Sahuguet. An electronic patient record "on steroids": Distributed, peer-to-peer, secure and privacy-conscious. In *VLDB*, pages 1273–1276, 2004.

[2] S. Abiteboul, B. Amann, J. Baumgarten, O. Benjelloun, F. Dang-Ngoc, and T. Milo. Schema-driven customization of Web services (demo). In *VLDB*, 2003.

[3] S. Abiteboul, O. Benjelloun, B. Cautis, I. Manolescu, T. Milo, and N. Preda. Lazy query evaluation for Active XML. In *ACM SIGMOD*, pages 227–238, 2004.

[4] S. Abiteboul, O. Benjelloun, I. Manolescu, T. Milo, and R. Weber. Active XML: A data-centric perspective on Web services. In *Web Dynamics*, pages 275–300, 2004.

[5] S. Abiteboul, O. Benjelloun, and T. Milo. Positive Active XML. In *ACM PODS*, pages 35–45, 2004.

[6] S. Abiteboul, O. Benjelloun, and T. Milo. The active xml project: an overview. Gemo Tech. Report 331, 2005.

[7] S. Abiteboul, A. Bonifati, G. Cobena, I. Manolescu, and T. Milo. Dynamic XML documents with distribution and replication. In *ACM SIGMOD*, 2003.

[8] S. Abiteboul, I. Manolescu, and N. Preda. Constructing and querying a peer-to-peer warehouse of XML resources. In *Semantic Web and Databases Workshop*, 2004.

[9] S. Abiteboul, I. Manolescu, and E. Taropa. A framework for distributed XML data management. In *EDBT*, pages 1049–1058, 2006.

[10] S. Abiteboul, B. Nguyen, and G. Ruberg. *Building an Active Content Warehouse.*, chapter III, pages 63–95. Idea Group Publishing, 2006.

[11] A. V. Aho, J. E. Hopcroft, and J. D. Ullman. *Data Structures and Algorithms*. Addison-Wesley, 1983.

[12] M. N. Alpdemir, A. Mukherjee, A. Gounaris, N. Paton, P. Watson, A. Fernandes, and D. Fitzgerald. OGSA-DQP: A Service for Distributed Querying on the Grid. In *EDBT, LNCS 2992*, pages 858–861, 2004.

[13] ActiveXML home page. `http://www.activexml.net`.

[14] J. Blythe, E. Deelman, and Y. Gil. Automatically composed workflows for grid environments. *IEEE Intelligent Systems*, 19(4):16–23, 2004.

[15] J. Blythe, S. Jain, E. Deelman, A. Mandal, and K. Kennedy. Task Scheduling Strategies for Workflow-based Applications in Grids. In *IEEE Int. Symposium on Cluster Computing and Grid (CCGrid)*, 2005.

[16] L. Bouganim, F. Fabret, C. Mohan, and P. Valduriez. Dynamic query scheduling in data integration systems. In *ICDE*, pages 425–434, 2000.

[17] W. B. Bradley and D. P. Maher. The NEMO P2P service orchestration framework. In *HICSS*, 2004.

[18] T. D. Braun, H. J. Siegel, N. Beck, L. Bölöni, M. Maheswaran, A. I. Reuther, J. P. Robertson, M. D. Theys, B. Yao, D. A. Hensgen, and R. F. Freund. A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems. *J. Parallel Distrib. Comput.*, 61(6):810–837, 2001.

[19] M. H. Burstein, J. R. Hobbs, O. Lassila, D. L. Martin, D. V. McDermott, S. A. McIlraith, S. Narayanan, M. Paolucci, T. R. Payne, and K. P. Sycara. DAML-S: Web service description for the Semantic Web. In *International Semantic Web Conference*, pages 348–363, 2002.

[20] L. Cherkasova, Y. Fu, W. Tang, and A. Vahdat. Measuring and characterizing end-to-end Internet service performance. In *ACM Transactions on Internet Technology*, volume 3, 2003.

[21] W. K.-W. Cheung, J. Liu, K. H. Tsang, and R. K. Wong. Towards autonomous service composition in a grid environment. In *ICWS*, pages 550–557, 2004.

[22] R. Cole and G. Graefe. Optimization of dynamic query evaluation plans. In *ACM SIGMOD*, pages 150–160, 1994.

[23] L. A. G. da Costa, P. F. Pires, and M. Mattoso. Automatic composition of Web services with contingency plans. In *ICWS*, pages 454–461, 2004.

[24] E. Deelman, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, K. Blackburn, A. Lazzarini, A. Arbree, R. Cavanaugh, and S. Koranda. Mapping Abstract Complex Workflows onto Grid Environments. *J. Grid Comput.*, 1(1):25–39, 2003.

[25] A. Doan and A. Y. Halevy. Efficiently ordering query plans for data integration. In *ICDE*, pages 393–402, 2002.

[26] F. Ferreira, C. J. P. de Lucena, and D. Schwabe. A Peer-To-Peer platform based on Semantic Web Services. In *WWW (Posters)*, 2003.

[27] A. Gounaris, R. Sakellariou, N. W. Paton, and A. A. A. Fernandes. Resource scheduling for parallel query processing on computational Grids. In *GRID*, pages 396–401, 2004.

[28] J. M. Hellerstein, M. J. Franklin, S. Chandrasekaran, A. Deshpande, K. Hildrum, S. Madden, V. Raman, and M. A. Shah. Adaptive query processing: Technology in evolution. *IEEE Data Eng. Bull.*, 23(2):7–18, 2000.

[29] L. Huang, D. W. Walker, Y. Huang, and O. F. Rana. Dynamic Web service selection for workflow optimisation. In *AHM*, 2005.

[30] R. Hull, F. Llirbat, B. Kumar, G. Zhou, G. Dong, and J. Su. Optimization techniques for data-intensive decision flows. In *ICDE*, pages 281–292, 2000.

[31] JDSL - the data structures library in Java. `http://www.cs.brown.edu/cgc/jdsl/`.

[32] T. Jim and D. Suciu. Dynamically Distributed Query Evaluation. In *ACM PODS*, pages 413–424, 2001.

[33] Y.-K. Kwok and I. Ahmad. Static scheduling algorithms for allocating directed task graphs to multiprocessors. *ACM Comput. Surv.*, 31(4):406–471, 1999.

[34] L. Liu, C. Pu, and D. D. A. Ruiz. A systematic approach to flexible specification, composition, and restructuring of workflow activities. *J. Database Manag.*, 15(1):1–40, 2004.

[35] H. Ludwig, A. Keller, A. Dan, R. King, and R. Franck. Web Service Level Agreement (WSLA) Language Specification. IBM Software Group, Tech. Report, January 2003.

[36] B. Medjahed and A. Bouguettaya. A dynamic foundational architecture for semantic Web services. *Distributed and Parallel Databases*, 17(2):179–206, 2005.

[37] N. C. Mendonça and J. A. F. Silva. An empirical evaluation of client-side server selection policies for accessing replicated Web services. In *SAC*, pages 1704–1708, 2005.

[38] L. A. Meyer. *Estrategias para o Escalonamento Dinamico de Workflows em Grid. (in Portuguese)*. PhD thesis, COPPE/UFRJ, 2006.

[39] T. Milo, S. Abiteboul, B. Amann, O. Benjelloun, and F. D. Ngoc. Exchanging intensional XML data. In *ACM SIGMOD*, pages 289–300, 2003.

[40] W. Nejdl, M. Wolpers, W. Siberski, C. Schmitz, M. T. Schlosser, I. Brunkhorst, and A. Löser. Super-peer-based routing and clustering strategies for RDF-based peer-to-peer networks. In *WWW*, pages 536–543, 2003.

[41] OWL-S: Semantic Markup for Web Services. `http://www.w3.org/Submission/OWL-S/`.

[42] Pegasus home page. `http://pegasus.isi.edu`.

[43] R. Rajamony and M. Elnozahy. Measuring client-perceived response times on the WWW. In *"USENIX Symposium on Internet Technologies and Systems (USITS)*, 2003.

[44] N. Ruberg, G. Ruberg, and I. Manolescu. Towards cost-based optimization for data-intensive Web service computations. In *SBBD*, pages 283–297, 2004.

[45] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach (2nd Edition)*. Prentice Hall, 2003.

[46] R. Sakellariou and H. Zhao. A hybrid heuristic for DAG scheduling on heterogeneous systems. *IPDPS*, 2(2):111b, 2004.

[47] R. Sakellariou and H. Zhao. A low-cost rescheduling policy for efficient mapping of workflows on grid systems. *Scientific Programming*, 12(4):253–262, 2004.

[48] ACM SIGMOD Record articles database. Available at http://acm.org/sigmod/record/xml/.

[49] Simple Object Access Protocol (SOAP) 1.1. `http://www.w3.org/TR/SOAP`.

[50] D. Suciu. Distributed query evaluation on semistructured data. *ACM Trans. Database Syst.*, 27(1):1–62, 2002.

[51] C. Team. DAGMan (Directed Acyclic Graph Manager) meta-scheduler for condor. University of Wisconsin-Madison. `http://www.cs.wisc.edu/condor/dagman/`.

[52] Universal Description, Discovery, and Integration of Business for the Web (UDDI). `http://www.uddi.org`.

[53] T. Urhan, M. J. Franklin, and L. Amsaleg. Cost based query scrambling for initial delays. In *SIGMOD Conference*, pages 130–141, 1998.

[54] W. M. P. van der Aalst, A. H. M. ter Hofstede, B. Kiepuszewski, and A. P. Barros. Workflow patterns. *Distributed and Parallel Databases*, 14(1):5–51, 2003.

[55] P. K. Vargas, I. de Castro Dutra, and C. Geyer. Application partitioning and hierarchical application management in grid environments. Tech. Report ES-657/04, COPPE/UFRJ, 2004.

[56] P. K. Vargas, I. de Castro Dutra, V. Nascimento, L. Santos, L. Silva, C. Geyer, and B. Schulze. Hierarchical submission in a grid environment. In *MGC*, pages 1–6, 2005.

[57] K. Verma, K. Gomadam, A. P. Sheth, J. A. Miller, and Z. Wu. The METEOR-S approach for configuring and executing dynamic Web processes. LSDIS Tech. Report, University of Georgia, June 2005.

[58] D. W. The quintessential Linux benchmark: All about the BogoMips number displayed when Linux boots. *Linux Journal*, 21, 1996.

[59] The World Wide Web Consortium. `http://www.w3.org/`.

[60] The Web Services Activity Report. `http://www.w3.org/2002/ws`.

[61] M. Wieczorek, R. Prodan, and T. Fahringer. Scheduling of scientific workflows in the ASKALON grid environment. *SIGMOD Record*, 34(3):56–62, 2005.

[62] Web Services Definition Language (WSDL). `http://www.w3.org/TR/wsdl`.

[63] M.-Y. Wu, W. Shu, and J. Gu. Efficient local search for DAG scheduling. *IEEE Trans. Parallel Distrib. Syst.*, 12(6):617–627, 2001.

[64] X-OQL homepage. Available at http://activexml.net/xoql/.

[65] XML Path Language (XPath) Version 1.0. `http://www.w3.org/TR/xpath`.

[66] L. Zeng, B. Benatallah, A. H. H. Ngu, M. Dumas, J. Kalagnanam, and H. Chang. QoS-aware middleware for Web services composition. *IEEE Trans. Software Eng.*, 30(5):311–327, 2004.