GERAÇÃO EFICIENTE DE PLANOS DE MATERIALIZAÇÃO PARA DOCUMENTOS XML ATIVOS

Daniela Marques Pereira

DISSERTAÇÃO 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 MESTRE EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

Aprovada por:

Prof a . Marta Lima de Queirós Mattoso, D.Sc.

Prof. Gerson Zaverucha, Ph. D.

Prof. Angelo Roncalli Alencar Brayner, D.Sc.

RIO DE JANEIRO, RJ – BRASIL

ABRIL DE 2007

PEREIRA, DANIELA MARQUES

Geração Eficiente de Planos de Materialização para Documentos XML Ativos [Rio de Janeiro] 2007

XII, 103 p. 29,7 cm (COPPE/UFRJ, M.Sc., Engenharia de Sistemas e Computação, 2007)

Dissertação – Universidade Federal do Rio de Janeiro, COPPE

1. Materialização de Documentos AXML

I. COPPE/UFRJ II. Título (série)

A minha família,

"E o que importa não é o que você tem na vida, mas quem você tem na vida. E que bons amigos são a família que nos permitiram escolher."

Agradecimentos

Esta dissertação de mestrado não existiria se não fosse o apoio das pessoas que tanto prezo e amo. A estas agradeço pela compreensão, colaboração e pelo apoio prestado.

Meu sincero: muito obrigada!

À professora Marta Mattoso pelos conselhos, pelas dicas e orientações. Sempre atenciosa e prestativa, é uma pessoa que merece todo o respeito e admiração. Fico lisonjeada de tê-la tido como orientadora.

À Gabriela Ruberg. Sua incansável paciência, dedicação, atenção e colaboração foram imprescindíveis para a conclusão desta dissertação de Mestrado. Obrigada por todo o incentivo acadêmico e pelas maravilhosas idéias. Graças a ela e a sua presteza conseguimos apresentar, por dois anos consecutivos, nossos artigos no SBBD.

Aos membros da banca. É muito bom poder contar com suas observações, críticas e comentários. São sempre construtivos e nos ajudam a crescer e a aprender ainda mais.

Obrigada a todos da COPPE/UFRJ, principalmente aos funcionários e professores do Programa de Engenharia de Sistemas e Computação (PESC). Em especial, gostaria de agradecer ao professor Jano Moreira e Geraldo Xexéo pela minha aceitação no programa e pela confiança. Obrigada também pela oportunidade de trabalhar no projeto da Star One. Foi um grande aprendizado.

Ao CNPQ pela bolsa de estudos que me permitiu desenvolver e concluir esta dissertação.

À minha fiel amiga Aline. Às minhas primas Isabela e Luciana. À minha prima Céline. A vocês agradeço os ótimos momentos de descontração.

Aos amigos que conquistei no mestrado: Marcelo, Alexandre, Jairo, Rodrigo e Marisa. Além das muitas horas de estudo e dedicação, também tivemos muitos momentos divertidos!

Ao pessoal da Star One: Josué e Silvino pela compreensão nas faltas e pela confiança no Portal Lab, aos amigos Nelson, Cintia, Clarissa, Pedro, João e Carlos pela

amizade e companheirismo e ao meu grande amigo Rafael pela paciência, pelas discussões, pelos debates, pelos conselhos, pelas dicas e até pelos momentos filosóficos.

Ao pessoal da Universidade Católica de Petrópolis por sempre acreditarem em mim.

A minha família: tios, tias, primos, minha afilhada Natália e seu irmão Ricardinho por compreenderem minhas ausências.

A minha tão amada irmã Paula e Felipe por agüentarem com paciência os momentos de stress e me perdoarem pelas tantas vezes em que agi com rispidez.

Ao meu namorado Leonardo pelo incentivo, apoio, pelo conforto nos momentos difíceis e por entender os momentos de ausência ou falta de atenção.

Ao meu avô Manoel. Sempre bondoso, amável e atencioso. Muitas vezes não compreendia ao certo os motivos de minhas ausências, mas ainda assim me apoiava e dava forças.

A meus pais: Lucinda e Bernardino pela credibilidade, pelo "paitrocínio", pela força e incentivo. A vocês devo minha vida, minha carreira profissional e acadêmica, minhas conquistas. Não seria nada sem vocês ao meu lado.

A Deus que me deu saúde, uma família maravilhosa, amigos incríveis e me rodeou de pessoas competentes e permitiu assim que eu chegasse até o fim do mestrado, cumprindo com os meus objetivos e obrigações.

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

GERAÇÃO EFICIENTE DE PLANOS DE MATERIALIZAÇÃO PARA DOCUMENTOS XML ATIVOS

Daniela Marques Pereira

Abril/2007

Orientador: Marta Lima de Queirós Mattoso
Programa: Engenharia de Sistemas e Computação

Com o sucesso dos serviços Web e da linguagem XML como formatos padrões para a troca de informações entre aplicações, surgiu uma nova classe de documentos chamados de documentos XML ativos (AXML). Estes documentos possuem elementos intencionais que correspondem às chamadas de serviços Web, as quais podem ser executadas para que seus resultados sejam inseridos no documento. Portanto, para "materializar" o conteúdo completo de um documento AXML, é necessário executar todas as suas chamadas de serviços. Em ambientes distribuídos, serviços Web são geralmente providos por vários sítios e a execução de um serviço pode ser delegada a outros sítios. Assim, existem vários planos de materialização equivalentes. Além disso, podem existir dependências de execução em um plano de materialização, pois elementos intencionais podem estar arbitrariamente aninhados. Por conseqüência, gerar e escolher bons planos de materialização para um documento AXML é um problema difícil com complexidade exponencial. Tal complexidade inviabiliza o uso de métodos de busca exaustiva. Esta dissertação propõe, portanto a SLS-MC, uma estratégia de otimização baseada em busca local estocástica com múltiplas condições de parada, para a geração eficiente de planos de materialização. A SLS-MC foi projetada de forma a trabalhar em conjunto com uma estratégia global de otimização e foi implementada em um ambiente de simulação desenvolvido chamado SiMAX. Foram realizados vários experimentos que apontam o potencial de ganho de desempenho da SLS-MC.

Abstract of Dissertation presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Master of Science (M.Sc.)

EFFICIENT GENERATION OF MATERIALIZATION PLANS FOR ACTIVE XML DOCUMENTS

Daniela Marques Pereira

April/2007

Advisor: Marta Lima de Queirós Mattoso
Department: Computer Science and Systems Engineering

With the success of the Web services and the XML language as standards for information exchange among applications, a new class of documents, called AXML (Active XML) document, has emerged. These documents have special elements that correspond to Web service calls, which can be executed so that their results are inserted into the document. So, to materialize the whole content of an AXML document, it is necessary to execute all its Web service calls. In distributed environments, Web services are generally provided by many sites and the service calls execution may be delegated to other sites. This way, there are many equivalent materialization plans. Besides, there may also exist execution dependencies in a plan, as the intentional elements may be arbitrarily nested. In consequence, generating and choosing good materialization plans for an AXML document is a hard problem with exponential complexity. This complexity prevents the use of exhaustive search methods. Then, this dissertation proposes SLS-MC, an optimization strategy based on stochastic local search with multiple stop conditions, to help on the efficient generation of materialization plans. SLS-MC was designed to work with a global optimization strategy and was implemented in a simulated environment developed called SiMAX. Several conducted experiments show the SLS-MC potential to performance gains.

Índice do Texto

Índice de Figuras

Índice de Tabelas

Capítulo 1 – Introdução

Este capítulo apresenta o contexto e os objetivos abordados por esta dissertação de mestrado além de descrever sucintamente o escopo de cada um dos capítulos que compõem esta dissertação.

1.1 – Motivação

Em ambientes distribuídos, como as redes ponto a ponto (P2P) (MILOJICIC, KALOGERAKI et al., 2002; NG, OOI et al., 2003; TATARINOV, IVES et al., 2003) e as grades computacionais (GRIDs) (GRID COMPUTING, 2006), o uso da linguagem XML (BRAY, PAOLI et al., 2004; EXTENSIBLE MARKUP LANGUAGE (XML), 2006) e da tecnologia de serviços Web (WEB SERVICES ACTIVITY, 2006) têm se tornado cada vez mais comum e necessário. Tanto a linguagem XML, quanto os serviços Web são considerados padrões para a troca de informações e interoperabilidade entre as aplicações através da Web. Isso ocorre em função de características como heterogeneidade, autonomia e dinamismo comuns a estas aplicações.

Algumas das aplicações executadas em ambientes distribuídos tratam da integração de dados e da gerência de processos na Web. Estas aplicações geralmente envolvem sistemas heterogêneos, desenvolvidos em linguagens e plataformas diferentes que se comunicam por meio de serviços publicados na Web, ou seja, através de serviços Web. Através da linguagem XML e de seus padrões tais como SOAP (BOX, EHNEBUSKE et al., 2000), WSDL (CHRISTENSEN, CURBERA et al., 2001) e UDDI (UDDI.ORG, 2006), a tecnologia de serviços Web permite que informações e serviços sejam compartilhados pela Internet e acessados por seus clientes, sem quaisquer restrições de compatibilidade e principalmente sem a necessidade de intervenção humana. Os serviços Web constituem, portanto uma poderosa ferramenta.

Com o sucesso dos serviços Web e da linguagem XML, surgiu uma nova classe de documentos XML chamados de Documentos **XML Ativos** (ou simplesmente **documentos AXML**) (ABITEBOUL, BENJELLOUN.O. et al., 2004; ABITEBOUL, BONIFATI et al., 2003; MILO, ABITEBOUL et al., 2003a; MILO, ABITEBOUL et

al., 2003b). Documentos AXML são documentos XML que possuem conteúdo dinâmico. Eles combinam dados XML com chamadas a serviços Web, cujos resultados são inseridos no documento original. Uma grande vantagem dos documentos AXML é a possibilidade de manter atualizadas, em documentos XML, informações voláteis, como a temperatura atual de uma cidade ou um saldo bancário. Por serem documentos XML, os documentos AXML podem ser utilizados como meio de troca de informação e integração de dados.

Um documento AXML possui diversas formas de combinar serviços Web. É possível especificar chamadas de serviço que recebem como parâmetros de entrada tanto elementos do próprio documento como resultados de outras chamadas de serviços Web do documento. É possível determinar ainda que uma chamada deva ser executada imediatamente após a execução de uma outra chamada. Estes relacionamentos entre as chamadas de serviços Web geram dependências, que determinam a ordem em que os serviços devem ser chamados e executados. Uma vez executados, seus resultados são materializados no documento e colaboram para obtenção de um resultado comum desejado.

Atualmente, vários sistemas se baseiam no uso de serviços Web. Por exemplo, a arquitetura OGSA (TOWARDS OPEN GRID SERVICES ARQUITECTURE, 2006) explora serviços Web para apoiar aplicações em *grids* computacionais. No contexto de sistemas P2P, serviços Web constituem um elemento fundamental da plataforma Active XML (ABITEBOUL, BENJELLOUN.O. et al., 2002; ABITEBOUL, MANOLESCU et al., 2004), o principal sistema para a gerência de documentos AXML. Essa plataforma permite diversas possibilidades de uso de serviços Web, além da definição e execução dinâmica de consultas sobre bases de dados AXML.

As dependências de execução de serviços Web, característica comuns a estes sistemas, geralmente são representadas através de árvores de execução ou grafos direcionais acíclicos (DAGs – Dynamic Aciclic Graphs) (BLYTHE, JAIN et al., 1998; KWOK e AHMAD, 1999; SAKELLARIOU e ZHAO, 2004; WU, SHU et al., 2001), onde $G = (V, A)$. Nesta representação, V é o conjunto de vértices que representam os serviços Web a serem executados e A o conjunto de arestas que representam as dependências existentes entre os serviços Web.

Outra característica comum é que, como estas aplicações são executadas em ambientes distribuídos e dinâmicos, é possível encontrarmos vários sítios provendo um mesmo serviço Web. Estes serviços são ditos semanticamente equivalentes, dada a

similaridade semântica presente em suas funcionalidades (AZEVEDO, 2004). Nesses ambientes, também é comum que os sítios colaborem entre si, dividindo a responsabilidade de chamar os serviços Web. Portanto, é possível que haja a delegação de chamadas a serviços Web a outros sítios com o objetivo de aproveitar melhor os *links* de comunicação e de dividir a carga de trabalho.

De acordo com estas características, um mesmo serviço Web pode ser executado, ou seja, materializado por um dentre vários sítios executores possíveis e também pode ser delegado para que um dentre vários sítios possíveis realize a chamada ao serviço Web. Assim, um mesmo documento pode ser materializado de muitas maneiras, ou seja, através de muitos planos equivalentes de materialização. O número de combinações pode ser muito grande, crescendo exponencialmente em relação ao número de serviços Web demandados, o número de possíveis sítios delegados e de sítios executores.

Analisando o relacionamento entre as chamadas de serviços Web de um documento AXML podemos perceber a presença de alguns padrões de *workflows*, tal como os definidos em (AALST, HOFSTEDE et al., 2003). Assim, de maneira abrangente, um documento AXML pode ser comparado a um *workflow* de serviços Web. *Workflow* ou fluxo de processos é a automação total ou parcial de um conjunto de atividades interligadas, que coletivamente alcançam um objetivo de negócio (WFMC, 1999). Em conjunto com os serviços Web, a tecnologia de *workflow* é utilizada para coordenar as interações entre vários serviços Web, os quais de forma composta visam alcançar um resultado de negócio comum. Os serviços Web representam os passos lógicos que compõem o *workflow*, ou seja, os passos que devem ser executados para que o resultado possa ser atingido. Existem hoje diversas linguagens de modelagem de processos de negócio que permitem realizar a composição de serviços Web sob a forma de *workflows*. Exemplos destas linguagens são: WSCI (ARKIN, ASKARY et al., 2002), BPML (ARKIN e INTALIO, 2002), BPEL4WS (BEA, IBM et al., 2003; VIRDELL, 2003), XLANG (THATTE, 2001), WSFL (LEYMANN, 2001) e BPSS (BUSINESS PROCESS PROJECT TEAM, 2001).

Conforme mostrado em (BLYTHE, JAIN et al., 1998; KWOK e AHMAD, 1999; RUBERG, RUBERG et al., 2004), o escalonamento de processos a máquinas heterogêneas, modelados a partir de um DAG complexo é um problema NP-completo. Esse é o caso da distribuição de tarefas a recursos computacionais distribuídos, que freqüentemente acontece em aplicações paralelas e em GRIDs computacionais, assim

como é o caso da busca por bons planos de execução para *workflows* e conseqüentemente da busca por bons planos de materialização para documentos AXML.

As aplicações distribuídas baseadas em serviços Web têm ainda em comum o fato de atuarem em ambientes extremamente dinâmicos, regidos por um grande número de variáveis que determinam a configuração e o comportamento destas aplicações. Exemplos desses ambientes são as redes P2P e os GRIDs. Um serviço pode, em um dado momento, não ser mais executado por um determinado sítio como outrora, por exemplo. Todo este dinamismo introduz mais dificuldades na escolha de um bom plano de materialização. Quando estes contemplam ainda grafos de dependências complexos, a possibilidade de delegação de tarefas e um conjunto específico de provedores para cada serviço Web como acontece, por exemplo, na materialização de documentos AXML, a procura por bons planos de materialização torna-se ainda mais complexa.

1.2 – Objetivos

De acordo com as características de execução de *workflows* baseados em serviços Web e as características dos ambientes distribuídos como redes P2P e GRIDs, um mesmo *workflow*, ou em nosso caso, um mesmo documento AXML pode ser materializado através de muitos planos equivalentes de materialização. Surge, então, o seguinte desafio: como escolher eficientemente um bom plano de materialização para um documento AXML baseado em serviços Web, levando em consideração o dinamismo dos ambientes distribuídos e o grande número de planos alternativos possíveis?

O tempo de otimização da materialização não pode ser muito demorado sob pena de o plano escolhido não ser mais válido no momento da execução. Considerando que o tamanho do espaço de busca tem influência direta no tempo de otimização, é preciso encontrar uma maneira de percorrê-lo de forma seletiva, evitando a análise de todas as alternativas.

Soluções clássicas utilizadas na otimização de planos de execução como técnicas exaustivas, gulosas, de ramificação e poda *(branch and bound),* como a busca A*, e algoritmos de programação dinâmica (CORMEN, LEISERSON et al., 2001; RUSSEL e NORVIG, 2003) não são diretamente aplicáveis na resolução deste problema. Como o número de possibilidades é muito grande, torna-se inviável analisar e comparar todas as possibilidades, como sugere a solução exaustiva. Soluções puramente gulosas também

não são adequadas, pois tomam decisões pontuais e "míopes" em relação ao restante do plano, podendo deixar de "enxergar" soluções de materialização de alguns elementos em conjunto. Como planos alternativos devem ser comparados através de métricas de custo não monotônicas, pois um ótimo local não garante um ótimo global, é preciso analisar planos inteiros e técnicas de ramificação e poda e de programação dinâmica também não se aplicam.

É preciso pensar, portanto em uma nova solução que gere seletivamente planos equivalentes, os analise e escolha um plano suficientemente bom (porém, não necessariamente o melhor plano), dentro de um tempo de otimização aceitável. Em função da complexidade do problema, heurísticas tornam-se obrigatórias. O fator de comparação para a escolha dos melhores planos é o custo de execução, ou seja, o tempo de execução dos planos (*makespan*).

Recentemente, foi proposto o XCraft (RUBERG e MATTOSO, 2005), um otimizador para a materialização de documentos AXML. O XCraft abrange um conjunto de técnicas que lidam principalmente com a complexidade e o dinamismo da materialização de documentos AXML. No XCraft, uma álgebra de operadores baseados em serviços Web permite uma avaliação incremental de planos de materialização, reduzindo o espaço de busca ao mesmo tempo que considera aspectos dinâmicos dos sítios envolvidos. Assim, o XCraft busca obter informações atualizadas sobre o status dos sítios e entregar resultados parciais mais rapidamente. O XCraft aborda uma estratégia iterativa que avalia dependências entre chamadas de serviços Web para quebrar um plano abstrato inicial em sub-planos menores. Cada sub-plano é otimizado localmente para gerar um sub-plano físico (*i.e.*, contendo os endereços dos sítios envolvidos), o qual é executado antes de prosseguir com a otimização dos demais sub-planos. A estratégia proposta no XCraft é bastante adequada para cenários P2P, pois permite uma melhor adaptação a mudanças no sistema. Além disso, embora dinâmica, essa estratégia considera fatores relevantes para a materialização de documentos AXML, como a heterogeneidade de um sistema P2P típico, execuções descentralizadas e a carga de processamento dos sítios. No entanto, existe uma dificuldade com relação à escolha do tamanho dos sub-planos. Enquanto tamanhos muito pequenos podem levar a resultados parecidos ao da busca puramente gulosa (que é míope em relação ao resto do plano), valores muito altos podem implicar tempos de otimização inaceitáveis, semelhantes aos da solução exaustiva. Um grave problema é que mesmo planos

pequenos podem ter tempos de otimização bastante grandes devido a um grande número de planos alternativos.

O objetivo desta dissertação de mestrado é contribuir com a solução do problema de materialização de documentos AXML. Nosso desafio é: como gerar e analisar bons planos de materialização em um tempo aceitável de otimização, sem que este tempo seja fortemente influenciado pelo tamanho do espaço de busca? Pretendemos obter bons planos de materialização tanto para sub-planos pequenos, quanto para sub-planos maiores, com um tempo de otimização sem muita flutuação, por exemplo. Para tal, é preciso percorrer seletivamente o espaço de busca, analisando e comparando planos inteiros segundo heurísticas pertinentes ao problema da materialização de documentos AXML. Para diminuir os efeitos da complexidade (*i.e.* número de planos alternativos possíveis) dos sub-planos, nossa solução tira proveito dos métodos de busca estocásticos.

A partir de um estudo sobre métodos de busca e problemas semelhantes à materialização de documentos AXML, nós propomos a **SLS-MC** (de "*Stochastic Local Search with Multiple stop Conditions*"), uma estratégia baseada em busca local estocástica com múltiplas condições de parada. Mesmo para grandes espaços de busca, a SLS-MC permite controlar o tempo gasto na otimização da materialização de documentos AXML e ainda assim gerar boas soluções. Conseqüentemente, ela reduz a sensibilidade do tempo de otimização ao tamanho do espaço de busca. Dessa forma, a SLS-MC representa uma alternativa atraente e inovadora dentre as soluções existentes para a materialização de documentos AXML.

Vale ressaltar, contudo, que a SLS-MC não é restrita ao contexto de documentos AXML, podendo ser aplicada na otimização de vários problemas semelhantes que envolvem a execução dinâmica e distribuída de processos participando de grandes espaços de soluções, como por exemplo, a otimização da execução de *workflows* baseados em serviços Web.

Problemas NP - completos semelhantes (BLUM e ROLI, 2003; KIRKPATRICK, GELATT et al., 1983; RUSSEL e NORVIG, 2003), cujo objetivo é também encontrar boas soluções em espaços de busca muito grandes, costumam utilizar soluções baseadas em algoritmos de busca local com condição de aceitação estocástica. Os algoritmos de busca local geram uma solução inicial, através de uma estratégia qualquer, que vai sendo refinada à medida que estados vizinhos vão sendo visitados. Soluções melhores são sempre aceitas. O algoritmo conhecido como Têmpera Simulada

(*Simulated Annealing*) (KIRKPATRICK, GELATT et al., 1983; RUSSEL e NORVIG, 2003) é um algoritmo de busca local estocástico que utiliza um fator de probabilidade para aceitação de planos piores, ou seja, segundo uma probabilidade p, estados vizinhos ruins são aceitos com o objetivo de escapar de possíveis mínimos locais. Quando a busca também se estabiliza, uma nova solução inicial é gerada de forma aleatória. A busca prossegue por um número determinado de vezes. A melhor solução de todas, encontrada durante a busca, é então escolhida.

Ao contrário da maioria dos algoritmos de busca local, a SLS-MC avalia múltiplas condições de parada para a busca por bons planos de materialização. Como o espaço de busca é muito grande neste problema, a busca não pode ser limitada a uma única condição de parada, como um número fixo de iterações, por exemplo. Esse tipo de condição de parada pode fazer com que o algoritmo de busca perca muito tempo tentando melhorar uma solução que já é suficientemente boa ou ainda, pode não prover oportunidades suficientes para encontrar uma solução mais adequada. A SLS-MC procura explorar diversas condições de parada que permitam ao algoritmo "perceber" quando uma boa solução já foi encontrada ou que ainda é preciso procurar um pouco mais. A SLS-MC permite ainda a utilização de diferentes heurísticas de agendamento de tarefas para a geração da solução inicial a partir da qual a busca local é realizada.

Esta abordagem é inovadora tanto no contexto dos documentos AXML como para *workflows* baseados em serviços Web. Ao contrário do que ocorre nas demais estratégias baseadas em busca local para *workflows* (BLYTHE, JAIN et al., 2005; WU, SHU et al., 2001), a SLS-MC considera que sítios podem colaborar através da delegação de tarefas e que cada tarefa do documento AXML possui um conjunto específico de provedores.

Para avaliar a SLS-MC, nós a aplicamos no processo de materialização de documentos AXML definido pelo otimizador XCraft (RUBERG e MATTOSO, 2005) da plataforma Active XML, onde o ambiente distribuído e dinâmico das redes P2P são a base de seu funcionamento. Em ambientes complexos como P2P, a realização de testes não é uma tarefa fácil. Assim, com o objetivo de facilitar a avaliação experimental de diferentes estratégias de materialização de documentos AXML, desenvolvemos o **SiMAX,** um simulador implementado a partir de um protótipo do otimizador XCraft (RUBERG e MATTOSO, 2005), que foi estendido com a estratégia SLS-MC.

O SiMAX permite a configuração de muitos cenários com diversos tipos de documentos AXML e redes P2P arbitrariamente complexas, além de diversas

estratégias de busca e heurísticas de agendamento de tarefas. Visando analisar o impacto desses fatores tanto no tempo gasto na otimização como no tempo de materialização do documento, foram conduzidos vários testes com o SiMAX envolvendo diferentes condições de parada e heurísticas de agendamento de tarefas.

Os testes realizados mostram que com um mesmo tempo de otimização, a SLS-MC consegue obter bons resultados para todos os diferentes documentos avaliados, independente de seu tamanho. A SLS-MC se mostrou mais eficiente que as soluções exaustiva e gulosa. Através das múltiplas condições de parada e das heurísticas de agendamento de tarefas utilizadas, a SLS-MC obteve tempos de otimização menores e custos de execução mais baixos, resultando em tempos totais inferiores aos obtidos pelas demais estratégias avaliadas. Os resultados evidenciam a contribuição da SLS-MC para a otimização da materialização de documentos AXML e conseqüentemente para a execução de *workflows* baseados em serviços Web.

1.3 – Organização do Documento

Esta dissertação de mestrado foi organizada em sete capítulos descritos a seguir.

Como os documentos AXML podem apresentar Grafos de Dependências bastante complexos e permitem ainda a delegação de chamadas a outros sítios e a escolha de um sítio executor dentre vários possíveis, esta classe de documentos necessita potencialmente de uma solução que escolha eficientemente um bom plano de materialização. Desta forma, o Capítulo 2 apresenta o conceito de documentos AXML, detalhando como estes se aplicam ao contexto de *workflows* baseados em serviços Web e quais são as necessidades para que documentos AXML possam ser materializados eficientemente. O Capítulo 2 apresenta ainda a plataforma Active XML, a principal ferramenta para a manipulação de documentos AXML e descreve o processo de materialização dos documentos AXML adotado pelo XCraft, um componente otimizador da plataforma Active XML.

O Capítulo 3 faz um levantamento sobre os principais métodos e técnicas de busca e otimização com o objetivo de contextualizar o problema e fundamentar a solução proposta: a SLS-MC, apresentada no Capítulo 4. A SLS-MC trata-se de uma solução menos sensível ao tamanho dos espaços de busca dos sub-planos e é complementar a solução proposta pelo otimizador XCraft. No Capítulo 4, além de uma descrição geral da solução, são apresentados os algoritmos utilizados na construção do

SiMAX, um simulador desenvolvido sobre a plataforma Active XML que foi estendida com a solução SLS-MC.

O Capítulo 5 apresenta o SiMAX que permite configurar diferentes estratégias de otimização e facilita de sobremaneira a realização de testes com a plataforma Active XML e documentos AXML.

Os resultados obtidos com a SLS-MC através dos testes realizados com o SiMAX são apresentados no Capítulo 6. Realizamos diversos testes, através dos quais procuramos avaliar diferentes condições de parada e também diferentes heurísticas para a geração da solução inicial. Sempre que possível os resultados foram comparados com os obtidos utilizando a solução exaustiva e gulosa.

Finalmente, o Capítulo 7, apresenta as conclusões obtidas através da realização desta dissertação, e deixa registradas sugestões de trabalhos futuros que possam gerar melhorias no processo de materialização de documentos AXML e também de *workflows* baseados em serviços Web.

Capítulo 2 – Documentos AXML e a Plataforma Active XML

Este capítulo apresenta o conceito de documentos AXML, descreve o processo de materialização e a necessidade de otimização destes documentos. Neste capítulo é apresentada ainda a arquitetura da plataforma Active XML, a principal ferramenta para a manipulação de documentos AXML. Os documentos AXML e a plataforma Active XML constituem-se na base fundamental sobre a qual a estratégia SLS-MC foi desenvolvida e avaliada. Ainda neste capítulo, é realizado um estudo sobre diversos trabalhos relacionados ao problema da materialização de documentos AXML que auxilia numa melhor compreensão do problema e na formulação da solução.

Com a evolução das redes de comunicação a da Internet, surgiram novas possibilidades de interação e comunicação entre pessoas e empresas. Hoje as empresas possuem escritórios e funcionários espalhados por todo o mundo, necessitando, portanto de meios eficientes de troca de informações através da Web. Da mesma maneira, as instituições de ensino vêem, através da Internet, a possibilidade de ampliação de pesquisa e de colaboração por meio da utilização de recursos dispersos. Como conseqüência, as aplicações de hoje necessitam trocar informações entre si para se manterem atualizadas e a atenderem às pessoas e às instituições espalhadas mundialmente.

Neste contexto da Web, a integração de dados se depara com novos desafios, particularmente em função da heterogeneidade e não interoperabilidade das fontes de dados e aplicações. Estes aspectos foram parcialmente sanados com a criação de padrões que procuravam tornar a comunicação viável, independente da tecnologia utilizada (ABITEBOUL, BENJELLOUN.O. et al., 2002). Um destes padrões é a linguagem XML (*Extensible Markup Language*), uma linguagem de marcação que permite integrar dados e metadados e que assumiu um papel muito importante na troca de dados e informações pela Web (XML, 2006). O advento da linguagem XML também tornou possível o desenvolvimento de serviços integráveis e independentes de plataforma devido ao alto grau de interoperabilidade obtido, também, pelo uso de

padrões, tais como SOAP e WSDL. Estes serviços são conhecidos como Serviços Web (ou *Web Services*). O objetivo dos serviços Web é explorar a tecnologia XML e a Internet para integrar aplicações que possam ser publicadas, localizadas e invocadas na Web. Por meio do uso de padrões, a comunicação entre as aplicações se da de forma independente de sistema, linguagem de programação ou plataforma.

O grande poder da linguagem XML e dos serviços Web logo ficou evidente e hoje ambos são adotados como padrões para a troca de informações e comunicação entre aplicações através da Web. No entanto, nem sempre é possível realizar a interação entre aplicações utilizando apenas padrões de mensagens e protocolos. Muitas vezes, as aplicações necessitam de interações mais complexas, que consomem bastante tempo e requerem serviços mais elaborados (AALST, 2003). Sob esta motivação surgiram os documentos AXML que consistem em documentos XML dinâmicos que combinam dados XML com a execução de serviços Web. Por meio da plataforma Active XML, uma plataforma P2P para a manipulação de documentos AXML, é possível trocar dados AXML pela Web.

Este capítulo descreve os documentos AXML e a plataforma Active XML. Ele descreve ainda o processo de materialização de documentos AXML abordado pelo XCraft, o otimizador da plataforma Active XML, que serve de base para a aplicação da SLS-MC.

2.1 – Documentos AXML

Documentos AXML são uma nova geração de documentos XML nos quais algumas informações são definidas explicitamente e outras definidas intencionalmente por meio de chamadas a serviços Web. Estas chamadas são representadas através de elementos especiais contidos no documento XML, que além de informarem o serviço Web a ser executado, passam algumas outras informações como o tempo de validade dos resultados e se estes devem substituir a chamada ou se devem ser agregados a mesma (MILO, ABITEBOUL et al., 2003a). Juntos, documentos XML e serviços Web constituem em um poderoso *framework* para o gerenciamento de dados distribuídos (ABITEBOUL, BENJELLOUN.O. et al., 2004) cujo principal objetivo é integrar informações providas por um número autônomo de sítios heterogêneos e poder consultá-las de forma uniforme.

Uma vantagem dos dados AXML é a possibilidade de se manterem atualizadas em documentos XML as informações dinâmicas geradas por meio de chamadas a serviços Web capazes de produzirem as informações demandadas pelo documento. Os documentos AXML são dinâmicos, pois é possível determinar quando as chamadas devem ser disparadas e por quanto tempo seus resultados são válidos (ABITEBOUL, BENJELLOUN.O. et al., 2002). Por exemplo: uma chamada pode ser configurada para ser disparada de tanto em tanto tempo, ou apenas quando as chamadas que dela dependem forem invocadas, gerando assim, uma materialização em cadeia.

Após a execução dos serviços Web, os resultados obtidos são materializados no documento atualizando seu conteúdo. Isso permite que os documentos AXML evoluam ao longo do tempo, como um resultado de suas chamadas de serviço. Desta maneira, as chamadas de serviço constituem dados intencionais que simbolizam os dados resultantes da execução de serviços Web. Logo, para materializar o conteúdo completo de um documento AXML, é necessário executar todas as suas chamadas de serviço.

As chamadas a serviços Web podem precisar de parâmetros de entrada. Estes parâmetros podem corresponder a elementos do próprio documento, inclusive a outras chamadas de serviço. Além disso, as chamadas de serviço podem trazer como resultado da execução de seus serviços Web outras chamadas de serviço Web. Estas chamadas resultantes deverão ser igualmente executadas e seus resultados materializados no documento. Estas características tornam os documentos AXML ainda mais dinâmicos e adaptáveis.

A Figura 1 mostra o exemplo de um documento AXML (com notação simplificada para fins de clareza) contendo uma chamada de serviço Web representada pelo elemento "sc". O serviço referenciado chama-se *getTemperature*, o qual obtém a temperatura atual de uma dada cidade. O nome da cidade, cuja temperatura deve ser recuperada, deve ser passada à chamada de serviço por meio de um parâmetro, que pode ser um texto simples, um elemento XML ou uma consulta XPath a ser realizada sobre o próprio documento. No exemplo da Figura 1, a cidade é passada por meio do elemento *city*. Repare que o elemento *city* está contido no elemento *sc* e por isso constitui-se no parâmetro da chamada ao serviço *getTemperature*. Após a execução da chamada de serviço, o resultado é devidamente inserido no documento, eventualmente substituindo o elemento dinâmico, se assim for especificado pelo usuário.

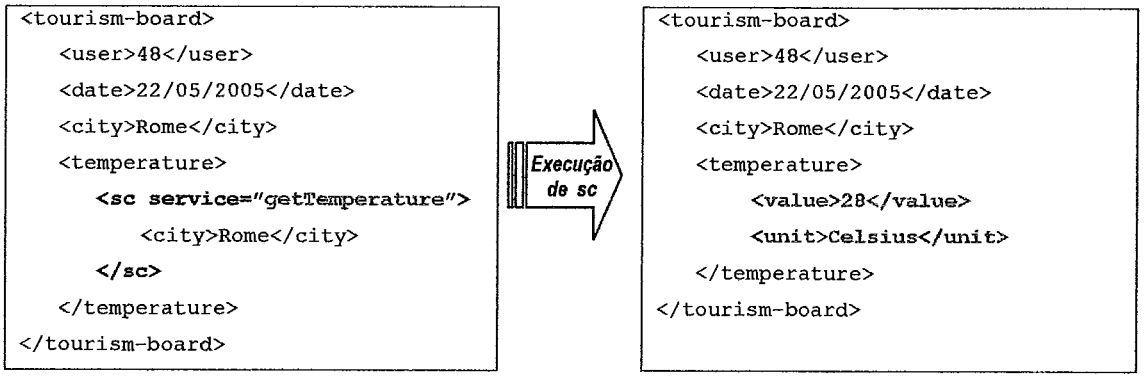


Figura 1 - Evolução de um documento AXML após a execução de uma chamada de serviço Web

Neste exemplo a informação sobre a temperatura da cidade Roma estava representada implicitamente por meio da chamada ao serviço Web "*getTemperature*". Por estar usando dados intencionais, e não estáticos, o documento XML estará sempre disponibilizando dados atualizados sobre a temperatura da cidade Roma. Se houver, no entanto, o interesse em variar a cidade sobre a qual se deseja obter a temperatura, é possível tornar variável a informação *city*. Para tal, basta substituir *city* por uma chamada a um serviço Web que retorne nomes de cidade (segundo um critério qualquer).

A Figura 2 apresenta um exemplo de aninhamento de chamadas. O parâmetro de entrada do serviço *getTemperature* agora é definido implicitamente, por meio de uma chamada ao serviço Web *getCity*.

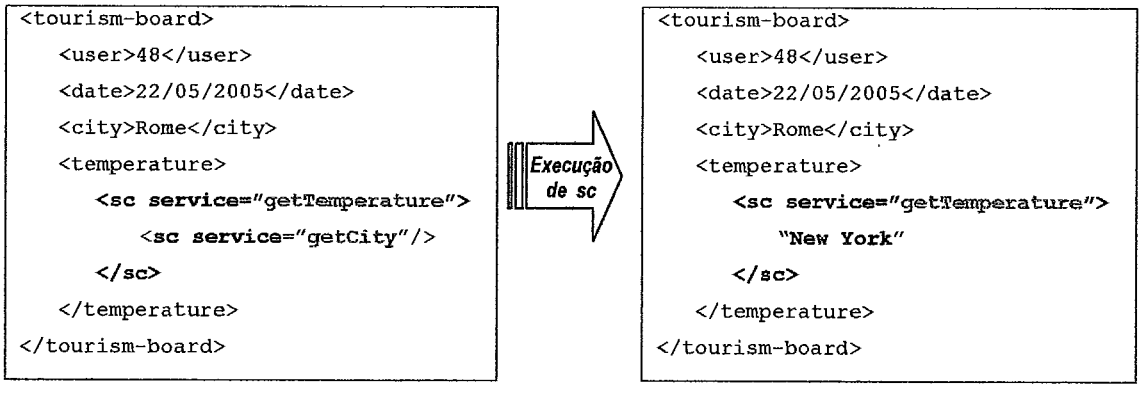


Figura 2 - Chamadas de serviços Web aninhados

2.2 – Plataforma Active XML

Para dar suporte aos documentos AXML, permitindo que estes possam ser armazenados, materializados e manipulados através da Web foi desenvolvida a

plataforma Active XML. A plataforma Active XML foi criada para ser a "cola" necessária para unir a linguagem XML, os serviços Web e a tecnologia P2P, que com sua arquitetura descentralizada e heterogênea está se tornando cada vez mais popular entre as aplicações de integração de dados (ABITEBOUL, BENJELLOUN.O. et al., 2002). Assim, a plataforma Active XML pode ser vista com um *framework* P2P de software livre para a manipulação de documentos AXML, que permite tanto o uso arbitrário de serviços Web baseados em SOAP (BOX, EHNEBUSKE et al., 2000), como também de serviços mais elaborados que fazem consultas e atualizações em documentos AXML.

Na plataforma Active XML os sítios responsáveis por armazenar e compartilhar documentos AXML são chamados de sítios AXML (*AXML peers).* Cada sítio AXML da rede P2P pode atuar tanto como consumidor, quanto como fornecedor de dados AXML e de serviços Web. Os sítios agem como clientes, pois as chamadas contidas em seus documentos utilizam serviços Web providos por outros sítios (AXML ou não). Ao mesmo tempo, estes sítios agem como servidores, pois fornecem, por meio de facilidades de consulta sobre seu repositório de documentos, serviços Web (Figura 3). Um sítio AXML é, portanto um integrador de dados e serviços: eles podem integrar dados XML obtidos de várias origens através de chamadas de serviço e também provedores de serviços: os serviços Web definidos por eles, podem ser demandados por outros sítios AXML, bem como por outros sítios que ignoram AXML e que entendem simplesmente SOAP (ABITEBOUL, BENJELLOUN.O. et al., 2002).

A arquitetura da plataforma AXML é apresentada na Figura 3.

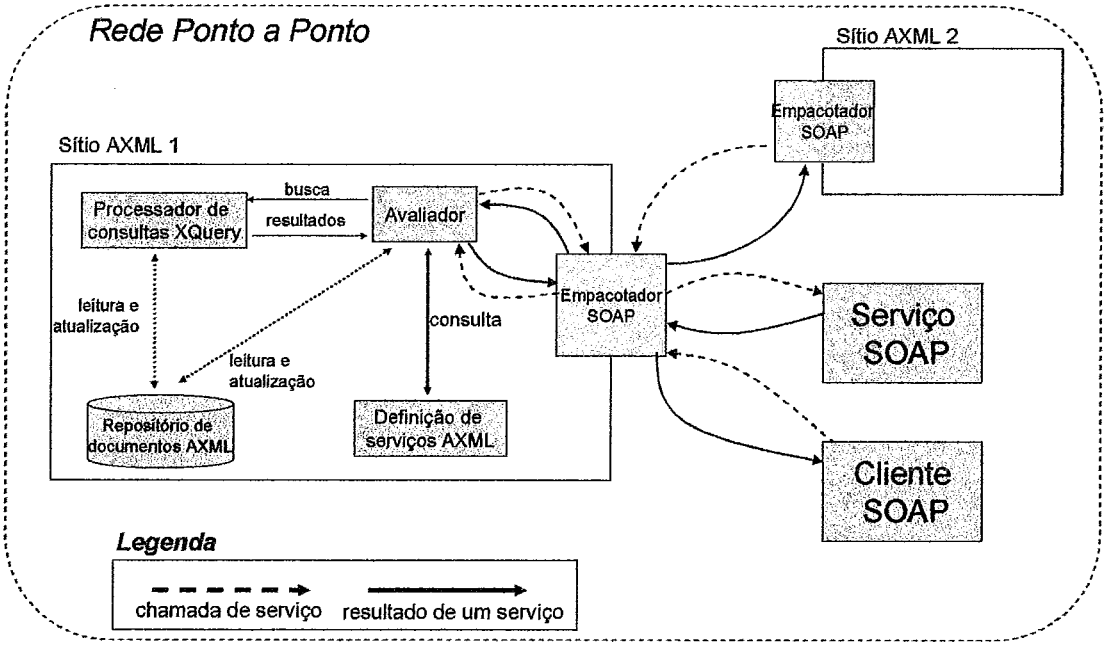


Figura 3 - Arquitetura da Plataforma AXML

Um sítio AXML recebe e envia requisições de serviços Web a outros sítios seguindo o padrão SOAP. O empacotamento e desempacotamento de mensagens SOAP são realizados pelo componente **Empacotador SOAP**. O componente **Avaliador** é responsável por executar as chamadas de serviço embutidas nos documentos AXML armazenados no **Repositório de Documentos AXML** do sítio AMXL. Chamadas a serviços locais são processadas localmente. Chamadas a serviços remotos são empacotadas e enviadas ao sítio provedor do serviço pelo Empacotador SOAP. Os resultados das chamadas, quando retornados, são desempacotados no Empacotador SOAP, recebidos pelo Avaliador e inseridos nos documentos AXML, enriquecendo-os.

As chamadas a serviços Web podem ser destinadas a sítios provedores de serviços que não seguem a plataforma AXML. Basta que estes falem SOAP. Do mesmo modo, outros clientes, que não sítios AXML, podem demandar os serviços Web providos por um sítio AXML. Os serviços Web fornecidos por um sítio AXML são definidos por meio de consultas e atualizações declarativas e parametrizadas realizadas sobre os documentos AXML armazenados em seu **Repositório de Documentos**. Estes serviços são definidos no componente **Definição de Serviços Web** e são expostos aos outros sítios por meio da interface regular WSDL (RUBERG, RUBERG et al., 2004).

O componente **Processador de Consultas XQuery** é o componente que permite que consultas XPath possam ser executadas sobre os documentos AXML e utilizadas como parâmetros de entrada para chamadas de serviço do mesmo documento. Ou seja, é possível definir um parâmetro de entrada de uma chamada de serviço de um documento AXML através de consultas XPath, que trazem como resultado elementos do próprio documento.

Hoje, a plataforma Active XML disponibilizada para a realização de testes segue a arquitetura da Figura 3, mas necessita que o usuário informe os sítios que devem executar os serviços Web demandados a partir dos documentos AXML. Hoje também não existe a possibilidade de delegação de chamadas e todos os serviços Web são sempre chamados a partir do sítio AXML que contém o documento a ser materializado. Esta situação é bem desconfortável e sujeita a falhas. É ainda pouco eficiente, pois diante das muitas possibilidades de execução, é possível que o usuário nem sempre escolha uma combinação de sítios provedores que seja capaz de tirar proveito dos *links* de comunicação e se adaptar a dinâmica das redes P2P. Sem a possibilidade de delegação de chamadas, a materialização de documentos AXML fica ainda mais ineficiente, podendo sofrer com o gargalo criado pelas chamadas em seqüência dos serviços. Em (ABITEBOUL, BONIFATI et al., 2003) é considerada a replicação de serviços Web e é proposta uma estratégia baseada em custo para escolher a melhor réplica. No entanto, a delegação de execuções ainda não é considerada e presume-se uma busca exaustiva.

Visando melhorar estes aspectos, permitindo que documentos complexos, com muitas chamadas de serviços pudessem ser definidos e materializados de forma fácil e eficiente, um componente otimizador, chamado XCraft (RUBERG e MATTOSO, 2005) foi recentemente desenvolvido e agregado à arquitetura da plataforma Active XML.

2.3 – XCraft – Otimizador da Plataforma Active XML

O otimizador XCraft foi desenvolvido com o objetivo de fornecer à plataforma Active XML uma estratégia de otimização que tire do usuário todo o trabalho de definir manualmente documentos AXML complexos e que escolha eficientemente uma boa alternativa de materialização dentre as muitas existentes.

A estratégia de otimização proposta pelo XCraft é dinâmica e procura abordar a invocação de serviços Web segundo as flutuações dos ambientes P2P. Por meio da

heurística dividir para conquistar, que analisa pontos disjuntos do grafo de forma independente, e da heurística de contexto, que procura evitar comunicações desnecessárias, limitando a escolha dos sítios executores dos serviços Web, o XCraft, conseguiu obter bons resultados em relação a uma solução puramente exaustiva. A estratégia é baseada em funções de custo e procura intercalar planejamento e execução. Estas características permitem ao XCraft reduzir o espaço de busca, obter informações mais atualizadas sobre o estado dos sítios e devolver resultados parciais ao usuário o quanto antes. Com o auxílio de um componente localizador de serviços o XCraft avalia os possíveis executores e delegados para cada chamada de serviço e tenta escolher, dentre as possíveis alternativas de materialização, uma suficientemente boa.

Para desenvolver o XCraft, seus desenvolvedores precisaram formalizar o problema da materialização de documentos AXML. Assim, propuseram uma forma canônica de representar as restrições de invocação dos serviços Web e um modelo de custo para avaliação dos planos equivalentes. Para compreender melhor o problema da materialização de documentos AXML e o funcionamento do XCraft, que motivam o desenvolvimento da estratégia SLS-MC, descrevemos a seguir o processo de materialização de documentos AXML tal como proposto em (RUBERG e MATTOSO, 2005) e utilizado no XCraft.

2.3.1 – Grafo de Dependências e Planos de Materialização de Documentos AXML

O processo de materialização de documentos AXML começa sempre no sítio AXML que hospeda o documento sendo materializado. Este sítio é chamado de **sítio mestre**. O sítio mestre deve identificar as **restrições** ou **dependências** existentes entre as chamadas de serviço e respeitar a ordem de execução. Existem dois tipos de restrições entre chamadas de serviço: **restrições de invocação** e **restrições de conseqüência**. As restrições de invocação ocorrem quando uma chamada de serviço consome os resultados fornecidos por outra chamada de serviço que precisa, portanto ser executada primeiro. Esta restrição pode ocorrer por meio de dois tipos de parâmetros: **concretos** e **não concretos**. Os parâmetros concretos são decorrentes do aninhamento de elementos intencionais *sc* (i.e., quando uma chamada de serviço recebe como parâmetros de entrada os resultados de outras chamadas). Os parâmetros não concretos são decorrentes de consultas XPath que selecionam outras chamadas de serviço ou elementos no documento sendo materializado.

As restrições de conseqüência são determinadas pelo atributo *followed by* dos elementos *sc* de um documento AXML. Este atributo recebe como valor uma chamada de serviço Web que deve ser imediatamente executada após a execução do elemento *sc* que define o atributo *followed by.*

O documento (resumido) da Figura 4 apresenta um exemplo de um documento AXML extraído de uma aplicação financeira (RUBERG e MATTOSO, 2005). Este documento apresenta todos os tipos de restrições mencionados. Neste exemplo, o elemento **sc**, com o atributo **id** igual a 1 (com sublinhado simples), tem um parâmetro concreto de entrada chamado *swaps* que corresponde a uma outra chamada de serviço, com atributo id igual a 2 (com sublinhado duplo). Esta chamada, por sua vez, também possui um parâmetro de entrada, não concreto, que corresponde ao resultado de uma consulta XPath, a ser realizada sobre o próprio documento. No exemplo, ela retornará o valor "XTecno Acme Ltd" que é o conteúdo do elemento para o qual a consulta aponta. A mesma consulta poderia apontar para um elemento que correspondesse a uma chamada de serviço e trazê-la como resultado, aumentando ainda mais as restrições existentes.

O mesmo documento apresenta um exemplo de restrição de conseqüência. O elemento sc com atributo id igual a 5 (com sublinhado tracejado), possui o atributo *followed by*, que indica que esta chamada de serviço deverá ser seguida da execução da chamada com id igual a 1.

```
<current_contract>
        <number>12345</number>
        <company>
                <name>XTchno Acme Ltd</name>
                <can_swap>
                        <sc id=1 service="CheckSwapStatus">
                                <param name="swaps">
                                        <sc id=2 service="GetCurrentSwaps">
                                                <xpath>//company/name</xpath>
                                        </sc>
                                </param>
                                                ...
                        </sc>
                </can_swap>
        </company>
                                                ...
        <swap_debt>
                <sc id=5 service="CalculateDebt" followed_by="1"/>
                                                ...
        </swap_debt>
                                                ...
</current_contract>
```

Figura 4 - Documento AXML com restrições de materialização

O ponto de partida da materialização de documentos AXML é a identificação das dependências entre as chamadas de serviço. Essa verificação é trivial para chamadas aninhadas, ou seja, para parâmetros concretos, mas requer uma análise mais sofisticada para detectar dependências decorrentes de parâmetros não-concretos. Os relacionamentos entre as chamadas de serviço podem ser bem complexos e impossíveis de serem representadas por uma árvore. Por isso, as dependências entre as chamadas são representadas por meio de grafos, que servem de base para o processo de materialização. Assim, de acordo com (RUBERG e MATTOSO, 2005; RUBERG, RUBERG et al., 2004) o conjunto de dependências de um documento *D* é representado por um grafo de dependências direcionado *GD*, tal que nodos denotam chamadas de serviço. Para qualquer par de nodos sci e scj em *GD*, existe uma aresta do nodo scj para sci denotada por scj→sci sempre que o resultado de scj (nó filho) for usado como parâmetro de entrada (concreto ou não) de sci (nó pai). As arestas representam as dependências existentes entre as chamadas de serviço e determinam a ordem em que devem ser executadas: primeiro os filhos e depois os pais. O XCraft considera apenas grafos válidos, ou seja, sem ciclos de dependência que resultem em algo do tipo sci→sci, com o objetivo de evitar a ocorrência de *loops* infinitos e *deadlocks*. Assim, o XCraft introduz uma definição geral para o grafo de dependências e depois restringe a validade deste para instancias acíclicas (RUBERG e MATTOSO, 2005).

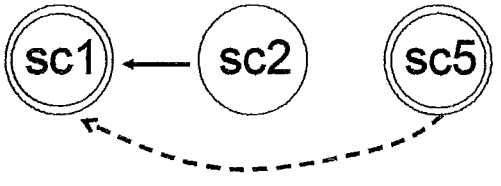


Figura 5 - Grafo de dependências

A Figura 5 apresenta o grafo de dependências do documento da Figura 4. Segundo o documento, a chamada com id igual a 2 precisa ser materializada primeiro para que a chamada com id igual a 1 possa então ser materializada. Este tipo de restrição é representado uma seta contínua. Já a chamada com id igual a 5 exige que tão logo ela seja materializada, a chamada com id 1 deve ser materializada. Esta restrição é representada por meio de uma seta tracejada. Os nodos com linhas duplas indicam que são nodos persistentes, ou seja, que seus resultados são sempre mantidos no documento, pois fazem parte do resultado final. Nodos com linhas simples são nodos auxiliares,

cujos resultados não precisam ser persistidos no documento, pois contribuem apenas para a obtenção dos resultados finais obtidos pelos nodos persistentes.

O número de dependências de uma chamada de serviço é conhecido como *fan-in* e é representado por *f*. Na Figura 5 a chamada com id igual a 1 tem f=2.

O plano de materialização de um documento *D* consiste basicamente no conjunto de árvores de expansão (*spanning trees*) extraídas a partir de *GD*, onde cada nodo está associado a um operador de avaliação da respectiva chamada de serviço. As árvores obtidas de *GD* são independentes entre si e podem ser analisadas e executadas individualmente e em paralelo. Existem dois tipos de planos de materialização: **planos abstratos** e **planos físicos**.

Os planos abstratos correspondem às árvores de expansão geradas a partir do GD. Estes planos possuem associados a seus nodos, operadores abstratos que determinam a ação de anotação a ser realizada para o nodo ao qual o operador corresponde. Dois operadores são utilizados nos planos abstratos: "μ", de *materialize* e "ρ", de *retrieve*. O operador "μ" indica de forma objetiva o conjunto dos possíveis executores para a chamada representada no nodo. Já o operador "ρ" indica que o sítio mestre não possui a informação sobre os possíveis executores, e que esta informação deve ser obtida por um outro sítio. Os operadores "ρ" são fortes indicadores de delegação.

Os planos abstratos podem ser encontrados em duas fases: **parcial** e **inicial**. O plano abstrato parcial possui "anotado" em cada nodo (quando o operador é do tipo "μ") o conjunto dos possíveis executores da chamada representada pelo nodo. O plano abstrato inicial possui, além destas informações, as informações dos possíveis delegados para a chamada. Os planos abstratos denotam o espaço de soluções possíveis para a materialização de um documento AXML, que pode ocorrer através de diferentes seqüências de delegação de chamadas e execução de serviços.

Os planos físicos possuem associados a seus nodos operadores que indicam explicitamente a tarefa que deve ser executada no processo de materialização. Para isso, os planos físicos possuem "anotados" em cada nodo: o sítio executor, que executará o serviço Web, e o sítio delegado, a partir do qual o serviço será demandado. O executor e o delegado foram escolhidos entre os conjuntos de possíveis delegados e executores associados aos nodos dos planos abstratos. Além disso, o plano físico determina a ordem de execução das chamadas.

Os operadores físicos mais comuns são: "I" de *invoke* e "δ" de *delegate*. O operador "I" determina que o nodo deve ser materializado através da invocação do serviço Web ao sítio anotado como executor. O operador "δ" determina que a sub-árvore, enraizada no nodo ao qual o operador está associado, deve ser encaminhada ao sítio delegado, para que lá a sub-árvore seja materializada segundo os operadores e informações anotadas. Os resultados finais devem ser retornados ao sitio de origem para que possam ser materializados no documento. Resultados intermediários podem ser descartados.

2.3.2 – Otimização e Materialização de Documentos AXML

Após a geração do gráfico de dependências e da extração das árvores de expansão, que são a base para os planos de materialização abstratos e físicos, é possível iniciar o processo pela escolha do plano físico que será utilizado para materializar o documento AXML. A Figura 6 mostra as principais etapas do processo de materialização de documentos AXML realizadas pelo XCraft e definidas em (RUBERG e MATTOSO, 2005).

Para cada árvore de expansão do documento (quadro 1), que representa uma tarefa independente de materialização, o sítio mestre usa as informações sobre os possíveis provedores de serviços, fornecidas pelo localizador de serviços, e gera o plano abstrato parcial (quadro 2). Neste plano, cada operador do plano é anotado com um conjunto de provedores, representado por Δe. Em seguida, o plano abstrato parcial é anotado com os possíveis delegados para cada chamada de serviço, gerando o plano abstrato inicial para o documento AXML. O conjunto dos possíveis delegados de cada chamada é representado por Δed (quadro 3 na Figura 6). Embora o sítio mestre possa delegar a execução de cada chamada de serviço a potencialmente qualquer outro sítio da rede, o processo aplica uma heurística de contexto que restringe os possíveis delegados a apenas sítios que possam executar alguma chamada de serviço contida no documento (RUBERG, RUBERG et al., 2004). O sítio mestre pode considerar, ainda, um conjunto conhecido de sítios que apresentem bom desempenho.

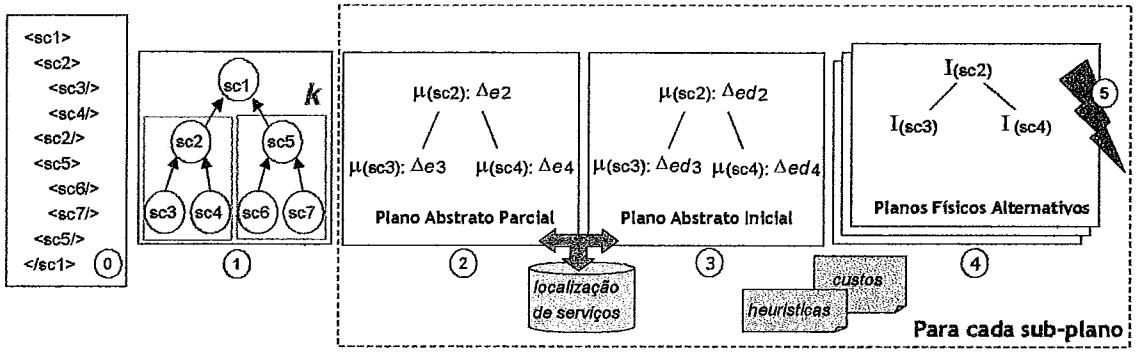


Figura 6 - Principais etapas da materialização de documentos AXML

A partir do plano abstrato inicial são gerados os possíveis planos físicos (quadro 4, na Figura 6), os quais são avaliados exaustivamente, segundo as métricas de custo adotadas. Após a comparação exaustiva dos planos equivalentes de materialização o melhor plano é escolhido e utilizado para materializar o documento AXML (Figura 6, etapa 5).

De acordo com (RUBERG, RUBERG et al., 2004), a complexidade de um documento AXML é dada em função do número total de planos físicos possíveis e pode ser definido da seguinte maneira: seja um documento D cujo grafo de dependências, GD, contém k árvores de expansão de altura h que são analisadas individualmente e onde cada chamada de serviço que possui dependência tem um *fan-in* f. Se cada serviço é provido por n sítios e cada chamada de D pode ser delegada a m possíveis sítios, o número de planos físicos possíveis é definido por Eq1:

$$\#\,physicalPlans(G_D) = k \times (n \times m)^x \qquad \text{Eq1}$$

O termo x da equação representa o aninhamento das chamadas de serviço e sua definição é dada em Eq2.

$$x = \sum_{i=0}^{h-1} f^i \qquad \text{Eq2}$$

As equações Eq1 e Eq2 indicam como o espaço de busca cresce rapidamente mesmo para valores baixos atribuídos às variáveis. Por exemplo, suponha que cada serviço Web do documento da Figura 6 (quadro 1) seja provido por 5 sítios alternativos,

sendo 15 sítios distintos os candidatos para a delegação de cada chamada de serviço. Deste modo temos 13.348.388.671.875 planos equivalentes para materializar o documento da Figura 6. Precisamos gerar um plano de materialização e para isso é preciso escolher um dentre os 13.348.388.671.875 planos possíveis. A otimização se faz necessária. É preciso analisar os planos alternativos da melhor maneira possível e escolher um plano eficiente de materialização de tal modo que este processo não impacte muito no tempo de resposta final do usuário. Usando uma solução exaustiva, por exemplo, levaríamos aproximadamente 215 anos para escolher o plano de materialização do documento da Figura 6 (assumindo-se que cada plano é analisado em 0,5 ms, tempo considerável para os computadores atuais). Este tempo é inadmissível.

A otimização de um documento AXML consiste em determinar quais sítios farão as invocações dos serviços e/ou quais sítios executarão cada serviço (levando em conta, por exemplo, os *links* de comunicação que ligam os sítios) de tal forma que as escolhas minimizem ao máximo o tempo de materialização do documento (*makespan).* Como o agendamento de tarefas em ambientes distribuídos (BRAUN, SIEGEL et al., 2001; KWOK e AHMAD, 1999), a otimização de documentos AXML também é um problema NP - completo, com complexidade exponencial em função do número de chamadas, de serviços Web e dos sítios participantes (RUBERG e MATTOSO, 2005). É preciso fazer uso de heurísticas.

Para comparar os planos e escolher o melhor, é necessário estimar o desempenho de cada plano alternativo gerado, baseando-se em funções que calculam o custo da execução de uma chamada de serviço Web. Diferentes métricas de custo podem ser usadas para guiar a busca por bons planos de materialização. As métricas definidas em (AZEVEDO, 2004), por exemplo, levam em consideração parâmetros de qualidade dos serviços Web. Em (CASANOVA, LEGRAND et al., 2000; SCHOPF e BERMAN, 1999) são propostas estratégias que usam a variabilidade das informações de performance (estimativas) com o objetivo de melhorar o desempenho geral das aplicações distribuídas e paralelas, uma vez que valores pontuais geralmente não são adequados para representar características que mudam ao longo do tempo. O XCraft utiliza as funções definidas em (RUBERG, RUBERG et al., 2004), que são próprias para serviços Web e levam em consideração o tempo de inicialização dos sítios, o tempo de execução do serviço Web, o tempo para transferir os dados pela rede e para inseri-los no documento. Essas funções são reproduzidas a seguir. O custo de execução

consiste basicamente no custo de execução do lado do cliente, que depende obviamente do custo do lado do servidor, onde o serviço é executado. O custo do lado cliente, representado por $ccost_{Pi}$, é calculado segundo a equação Eq3.

$$ccost_{Pi}(sc) = init_c + callSize \times pack_u + (callSize + envSize) \times net(1, P_i, P_j + scost\ P_j(sc) + resSize \times (unpack_u + parse(1, resSize) + merge(resSize, docSize(sc)) \quad \text{Eq3}$$

O cálculo de $ccost_{Pi}$ depende de $scost_{Pj}$, custo do lado do servidor, que deve ser calculado segundo a equação Eq4.

$$scost_{Pj} = init_a + callSize\ x\ (unpack_u + parse(1, callSize)) + srvExec + resSize \times pack_u + (resSize + envSize) \times net(1, P_j, P_i) \quad Eq4$$

As variáveis utilizadas em Eq3 e Eq4 são descritos na Tabela 1:

Tabela 1 - Variáveis para cálculo do custo de um plano de materialização (RUBERG, RUBERG et al., 2004)

Variáveis	Descrição
$init_c$ e $init_a$	Tempo para inicializar módulos respectivamente no cliente e no servidor
callSize e *resSize*	Tamanho (em bytes) da chamada de serviço e de seu resultado, respectivamente
Net (b, P_1, P_2)	Tempo médio para transferir b bytes de P_i para P_j
envSize	Tamanho médio (em bytes) do envelope SOAP
$pack_u$ e $unPack_u$	Tempo médio de empacotamento e desempacotamento SOAP de 1 byte de dados
docSize(sc)	Tamanho (em bytes) do documento que contem a chamada *sc*
parse(b,size)	Tempo médio para analisar b bytes de um fragmento AXML com *size* bytes
merge(b, size)	Tempo médio para juntar b bytes de dados em um documento AXML com *size* bytes
srvExec	Tempo de execução do servido no sítio executor

Além desses custos, é necessário também contabilizar o custo de delegação da sub-árvore enraizada na chamada de serviço, quando for o caso. Ao avaliarmos essas estimativas de custo nos planos físicos de materialização, percebemos que o custo de cada sub-árvore não pode ser calculado independentemente do resto do plano, pois o custo de delegação de um nodo depende da configuração de seus nodos ascendentes. Portanto, essas métricas de custo não são monotônicas, pois não preservam a ordem crescente ou decrescente de seus fatores e ótimos locais não levam a ótimos globais. É preciso, portanto avaliar planos inteiros.

Como cada uma das árvores de expansão pode ser ainda bem grande, resultando em espaços de busca enormes, o XCraft permite ainda, como parte de sua estratégia, quebrar cada uma das árvores de expansão em árvores ainda menores de altura k. Tal

como as árvores de expansão, cada sub-árvore gerada é também analisada de forma independente e executada logo em seguida a sua otimização. Igualmente, o XCraft consegue alternar otimização e execução, utilizando sempre informações atualizadas sobre a disponibilidade dos sítios da rede P2P, produzindo resultados mais rapidamente para o usuário. Da mesma maneira, por considerar apenas os sítios capazes de executar as tarefas contidas no plano de altura *k* sendo avaliado, ele consegue diminuir o espaço de busca e melhorar o tempo de otimização.

Apesar das melhorias introduzidas pelo XCraft, resta ainda uma dificuldade que precisa ser resolvida: a escolha da altura *k* dos sub-planos. Se a altura dos sub-planos for muito grande, o XCraft cai no problema da solução exaustiva que gasta muito tempo analisando todos os planos. Este tempo muitas vezes é proibitivo perante a dinâmica das redes P2P. Do mesmo modo, se os sub-planos forem muito pequenos, o XCraft deixa de avaliar alternativas que englobam um número maior de sítios e não percebem bons pontos de delegação. Por representarem uma visão muito limitada do plano geral estes planos pouco contribuem para a obtenção de um bom plano de materialização final. O desempenho acaba sendo semelhante ao de soluções gulosas. Além disso, mesmo sub-planos com pouca altura podem ser largos e bastante complexos, representando grandes espaços de busca. É preciso descobrir uma maneira de encontrar bons planos de materialização dentro de um tempo aceitável de otimização sem que este tempo seja influenciado pelo tamanho do espaço de busca dos sub-planos de altura *k*.

Assim, com o objetivo de apresentar uma alternativa menos sensível ao tamanho do espaço de busca dos sub-planos, propomos a SLS-MC, uma estratégia estocástica que acreditamos ser ainda mais eficiente principalmente se adotada em conjunto com a estratégia iterativa e incremental do XCraft.

2.4 – Trabalhos Relacionados

Os dados AXML não são novidade e podem ser encontrados em vários cenários. O processador Apache Jelly (APACHE JELLY, 2006), por exemplo, com o uso da linguagem Java, transforma dados XML em código executável. O Microsoft Office XP permite que documentos XML, utilizando *Smart Tags*, possam ser ligados a plataforma .NET para acesso a serviços Web. Existem também linguagens para integração de dados e serviços em plataformas P2P, como em (MILO, ABITEBOUL et al., 2003b). No entanto, a plataforma Active XML constitui-se na principal aplicação para a

manipulação e troca de documentos AXML, transformando uso de XML e serviços Web num poderoso e verdadeiro recurso para integração de dados (ABITEBOUL, BENJELLOUN.O. et al., 2002).

Em um sentido amplo, documentos AXML se assemelham às descrições de *workflows*, sendo caracterizados pela execução encadeada de serviços Web e troca de dados XML entre as invocações, como ocorre, por exemplo, na linguagem BPEL4WS (BEA, IBM et al., 2003). Documentos AXML podem compor e interconectar serviços Web externos providos por várias organizações distribuídas, com o objetivo de atingir um resultado de interesse de uma organização. No entanto, apesar das funcionalidades semelhantes, os documentos AMXL não são de fato *workflows*, já que eles não permitem a definição de fluxos de trabalho por meio de operadores condicionais e estruturas lógicas como acontece nas linguagens de definição de *workflows* conhecidas.

A materialização de documentos AXML pode ser vista como um problema de distribuição de tarefas dependentes a multi-processadores, tal como acontece em aplicações de processamento paralelo e distribuído ou grades computacionais. Nestas aplicações, o principal objetivo é agendar um grafo de dependências que contém n tarefas, *T1, T2 ... Tn* em p processadores ou recursos. Um agendamento *(schedule) S* pode ser definido como uma especificação da alocação das tarefas aos recursos. Para um agendamento S, a função $f(S)$ corresponde ao tempo de término geral. O objetivo do problema de distribuição de tarefas dependentes a multi-processadores é encontrar o agendamento S que minimize $f(S)$ (BELKHALE e BANERJEE, 1991). Apesar de a materialização de documentos AXML ter o mesmo objetivo que os problemas de distribuição de tarefas, existem diferenças sutis que impedem que as soluções normalmente utilizadas para resolver estes problemas sejam utilizadas. A principal razão é o fato de a materialização de documentos AXML considerar um conjunto definido e finito de recursos para a execução de cada serviço Web demandado e permitir a delegação de chamadas a serviços Web a outros sítios. Existe ainda o fato de os planos alternativos dos documentos AXML serem comparados por meio de funções de custo não monotônicas. A maioria das soluções para os problemas de distribuição de tarefas não aborda estas características.

Vários algoritmos já foram propostos para o agendamento de tarefas em ambientes com múltiplos recursos (KWOK e AHMAD, 1999). O algoritmo mais utilizado é o **agendamento de lista** *(list scheduling)* (KWOK e AHMAD, 1999;

SAKELLARIOU e ZHAO, 2004; WU, SHU et al., 2001), uma abordagem gulosa, que se baseia numa lista de prioridades de tarefas, onde os nós do grafo com maior prioridade são alocados primeiro. Os principais atributos utilizados na definição de prioridade das tarefas são os chamados nível-t e nível-b (*t-level e b-level*). O valor nível-t de um nó *ni* corresponde ao caminho mais longo desde um nó de entrada (ou seja, um nó sem pai) até o nó *ni*. Já o valor nível-b de um nó *ni*, corresponde ao caminho mais longo do nó *ni* (incluso) até um nó de saída (ou seja, sem filhos ou dependências). Os diferentes algoritmos de agendamento de lista utilizam os atributos nível-*t* e nível-*b* de formas diferentes. Alguns atribuem maior prioridade ao nó com menor nível-t enquanto outros atribuem maior prioridade ao nó com maior nível-b. Em geral, quando o agendamento é feito segundo o valor crescente de nível-t, o agendamento é feito em ordem topológica.

Outros algoritmos para o mapeamento de tarefas em recursos distribuídos surgiram, tais como: MCP, DSC, DLS, CPN entre outros (KWOK e AHMAD, 1999; WU, SHU et al., 2001). Cada um destes representa uma solução diferente de determinar a prioridade das tarefas usada no agendamento de lista. O algoritmo HEFT (WIECZOREK, PRODAN et al., 2005) trata-se de uma extensão desse clássico algoritmo para ambientes heterogêneos. É um algoritmo de três fases simples e viável que agenda *workflows* criando uma lista ordenada de tarefas e mapeia as tarefas aos recursos da forma mais apropriada. As três fases do algoritmo HEFT são: pontuação, classificação e mapeamento. A fase de pontuação determina os pesos dos nós e das arestas no *workflow*. A fase de classificação cria uma lista ordenada de tarefas, organizada na ordem em que eles deveriam ser executadas. Por fim, a fase de mapeamento assinala as tarefas aos recursos. Os pesos assinalados aos nós são calculados baseados nos tempos de execução previstos para as tarefas. Os pesos assinalados as arestas são calculados baseados nos tempos de transferências previstos entre os recursos. A fase de classificação é realizada percorrendo o DAG do *workflow* de baixo para cima e assinalando um valor de classificação para cada uma das tarefas. O valor de classificação é igual ao peso do nó mais o tempo de execução dos sucessores. Uma lista de recursos é organizada segundo os valores de classificação, em ordem decrescente. Na fase de mapeamento, tarefas consecutivas da lista de classificação são mapeadas aos recursos. Para cada tarefa, o recurso que prove o menor tempo esperado para terminar a execução da tarefa é escolhido.

Os algoritmos de lista são gulosos e abordam em sua maioria um agendamento estático, que é um agendamento feito em tempo de compilação (KWOK e AHMAD, 1999). Algoritmos gulosos não são adequados para a materialização de documentos AXML, pois tomam decisões pontuais e míopes, e o agendamento estático não é muito indicado para o ambiente dinâmico das redes P2P, pois decisões tomadas em tempo de compilação podem não ser mais válidas em tempo de execução. Algumas abordagens dinâmicas, como a DLS, tornam dinâmica a lista de prioridades. Em (SAKELLARIOU e ZHAO, 2004) é proposta uma solução híbrida do agendamento de lista, onde o propósito é utilizar a lista de prioridades para agrupar tarefas em grupos que possam ser alocados independentemente. Já em (WU, SHU et al., 2001) é apresentado o algoritmo TASK. Este algoritmo analisa as tarefas em ordem topológica, respeitando as dependências existentes e analisando primeiro os nós que fazem parte do caminho crítico, ou seja, os mais custosos. Ainda assim, estes algoritmos não são adequados, pois além de gulosos, e não compararem planos inteiros, não consideram a delegação de tarefas.

Os valores de nível-t e nível-b, utilizados para a determinação das prioridades na maioria das soluções enunciadas levam em conta os custos computacionais e de comunicação das tarefas. Geralmente estes valores são obtidos por estimativa, usando informações de perfil das operações tais como operações numéricas, operações de acesso de memória e primitivas de troca de mensagens. Em serviços Web, estas estimativas não são aplicáveis. Os serviços Web são vistos como caixas pretas, através das quais serviços são providos por meio de interfaces públicas. Serviços Web semanticamente equivalentes podem ser implementados internamente de formas totalmente diferentes e, portanto estas métricas não são adequadas à materialização de documentos AXML.

Um algoritmo interessante, que foge da lista de prioridades do agendamento de lista, é o Agendamento e Alocação Bolha - AAB (*Bubble Scheduling and Allocation*) (KWOK e AHMAD, 1999; OLIVER SINNEN e LEONEL SOUSA, 2001), que se baseia numa lista formada pelos nós do caminho critico e em uma lista de processadores. Uma solução inicial é gerada alocando-se todos os nós ao processador com o número maior de nós adjacentes. Em cada fase, o próximo processador da lista, é tido como o processador de referência e os nós alocados a eles são migrados para os processadores adjacentes. Se a migração conduzir à redução do tempo de execução do plano, a migração é acatada.

A submissão de tarefas em grades computacionais ou GRIDS também se assemelha ao problema da materialização de documentos AXML. As diferenças são pequenas, porém significativas. O objetivo principal de um GRID (GRID COMPUTING, 2006) é prover poder computacional, através de recursos heterogêneos distribuídos globalmente, a aplicações cujo objetivo é a resolução de problemas bastante complexos. Para a resolução destes problemas, as aplicações quebram o problema em tarefas menores, e as submetem aos recursos disponíveis visando distribuir a carga de processamento. Um dos grandes desafios nos sistemas de GRID é a necessidade de automação da distribuição, de gerenciamento e monitoramento das tarefas. Vários trabalhos foram propostos na tentativa de resolver estes problemas (MEYER, 2006; VARGAS, SANTOS.L.A.S. et al., 2005).

Na maioria das aplicações GRID, as dependências entre as tarefas também são representadas através de *DAGs*, mas as dependências são mais fracamente acopladas, representadas pela necessidade de utilização de arquivos de entrada produzidas como resultado por outras tarefas. Algumas soluções até abordam a delegação (VARGAS, SANTOS.L.A.S. et al., 2005), no entanto, para um número fixo de máquinas designadas exclusivamente para tal fim.

Tal como nas aplicações de processamento paralelo e distribuído, as aplicações GRID, em geral, consideram um pool de recursos e todas as tarefas podem ser mapeadas para qualquer uma deles. O objetivo maior é realizar um balanceamento de carga e não alocar as tarefas aos sítios específicos capazes de resolver a tarefa. O grande esforço está em encaminhar as tarefas à qualquer máquina capaz de executa-las naquele instante, evitando ao máximo que as máquinas fiquem ociosas. Em aplicações de processamento paralelo, é possível inclusive alocar uma mesma tarefa em dois recursos distintos com o objetivo de diminuir ainda mais o tempo de execução. Estes princípios não se aplicam aos serviços Web. Apenas recursos específicos podem executar um serviço Web e estes não podem ser quebrados e executados paralelamente.

Em (BLYTHE, JAIN et al., 1998) é apresentada uma solução para a alocação de recursos GRID, chamada de Abordagem Baseada em *Workflow* (ABW) (*Workflow Based Approach*). Esta abordagem também procura refinar uma solução inicial e utiliza algoritmos de busca local baseado em PBAGR (ou GRASP de *Greed Randomized Adaptative Search*). De forma semelhante à SLS-MC, a ABW procura evoluir a partir da solução inicial gerada de forma gulosa e cria e compara vários planos inteiramente. A ABW é executada durante um tempo limitado e enquanto existem tarefas disponíveis,

ela avalia o grau de aumento do tempo de execução para cada par tarefa-recurso. Aqueles pares que ficarem dentro de um determinado limite, são selecionados e uma deles escolhido aleatoriamente para ser mapeado. Após o mapeamento, o algoritmo calcula o momento no qual o recurso poderá executar a tarefa. Somente planos melhores são aceitos. Trata-se da solução mais parecida com a SLS-MC. No entanto, esta abordagem também não considera possíveis delegações e como mostram os experimentos não é adequada para *workflows* baseados em milhares de tarefas.

Em (YU e BUYYA, 2005) é realizada uma pesquisa sobre os principais sistemas gerenciadores de *workflows* para GRIDS que são avaliados segundo uma taxonomia para classificar e caracterizar as diversas abordagens para a construção e execução de *workflows* também apresentada neste trabalho. De acordo com a pesquisa apenas três sistemas apresentam uma arquitetura de agendamento descentralizada, através da qual os sistemas compreendem algum tipo de delegação de tarefas. Os três sistemas são: Triana, Askalon e GridBus Workflow. Apesar da arquitetura descentralizada, todos se baseiam em estratégias oportunistas com tomadas de decisão locais. O Askalon (WIECZOREK, PRODAN et al., 2005) se baseia no algoritmo HEFT que apesar de considerar e obedecer as dependências do plano (global), toma decisões locais na hora de alocar as tarefas aos recursos que provêem os tempos de término mais baixos. O Triana (TAYLOR, SHIELDS et al., 2003) é um ambiente gráfico de recursos baseados em GRID para a resolução e a composição de problemas. Este ambiente provê um portal que permite a composição de aplicações científicas baseadas nos princípios dos *workflows,* onde os usuários podem compor aplicações arrastando componentes de programação das barras de ferramentas para uma área de trabalho. Além de também tomar decisões locais na escolha dos recursos para executar as tarefas, cabe ao usuário definir quais recursos podem ser utilizados para serem utilizados como possíveis delegados. Por fim, o GridBus Workflow (YU e BUYYA, 2005) é um sistema com arquitetura hierárquica de agendamento que ajuda a alocar eficientemente os recursos a diferentes usuários e aplicações baseadas em seus requisitos de qualidade de serviço. Como os demais sistemas, o GridBus toma decisões locais e oportunistas.

Recentemente, como já mencionado, foi proposto um otimizador dinâmico para a materialização de documentos AXML denominado XCraft (RUBERG e MATTOSO, 2005). Para driblar o problema das soluções puramente gulosas ou exaustivas e para tentar se adaptar a dinâmica das redes P2P, o XCraft adotou uma estratégia que divide o plano original em sub-planos menores de altura k. Esta estratégia visa a reduzir a

complexidade total do problema, realizando otimizações locais. Ela permite ainda uma melhor adaptação ao dinamismo dos ambientes P2P, já que produz resultados rapidamente, ainda de acordo com as configurações vigentes no momento da otimização.

Apesar de diretamente vinculada ao problema, a solução adotada pelo XCraft permanece sensível ao tamanho do espaço de busca de soluções mesmo para sub-planos considerados pequenos. Existe uma dificuldade com relação à escolha do tamanho do sub-plano, pois ele pode conduzir a tempos altos de otimização, que como já mencionado, são inadequados ao dinamismo das redes P2P.

Baseados e motivados por esta dificuldade, e diante da inexistência de soluções que se aplicam diretamente ao problema da materialização de documentos AXML, nós desenvolvemos a SLS-MC. Por meio da busca local estocástica, a SLS-MC permite minimizar os efeitos do tamanho do espaço de busca.

Capítulo 3 – Métodos de Busca

Este capítulo tem por objetivo realizar um estudo sobre os principais métodos de busca e otimização utilizados em problemas semelhantes ao da materialização de documentos AXML.

A materialização de documentos AXML pode ser vista como um problema de planejamento ou um problema de busca. Um problema de planejamento básico possui uma descrição inicial do problema, uma descrição parcial do objetivo e um conjunto de ações (algumas vezes chamadas de operadores) que mapeiam uma descrição parcial em outra. Uma solução é uma seqüência de ações que conduzem a descrição do problema inicial à descrição do objetivo. Esta solução é chamada de plano (FREUDER, NAREYEK et al., 2005). Alguns problemas podem ter vários planos válidos que satisfazem a descrição do objetivo e precisam portanto de otimização, isto é, de um otimizador capaz de escolher eficientemente dentre os planos possíveis, um que minimize a função de custo objetivo do problema.

Geralmente, um otimizador é visto com o um conjunto de três componentes: o **espaço de busca**, o **modelo de custo** e a **estratégia de busca**. O espaço de busca é o conjunto de planos alternativos de execução para representar o plano original. O espaço de busca é obtido pela aplicação de regras de transformação e o modelo de custo prevê o custo de um dado plano de execução (VALDURIEZ e ÖZSU, 2001). A estratégia de busca explora o espaço de busca e seleciona o melhor plano, utilizando o modelo de custo. Ela define os planos que serão analisados e a ordem do exame. Existem enumeras estratégias de busca, cada uma própria para o tipo de problema que visam sanar.

A Figura 7 apresenta o modelo clássico de otimização, aqui aplicado no contexto de otimização de consultas distribuídas.

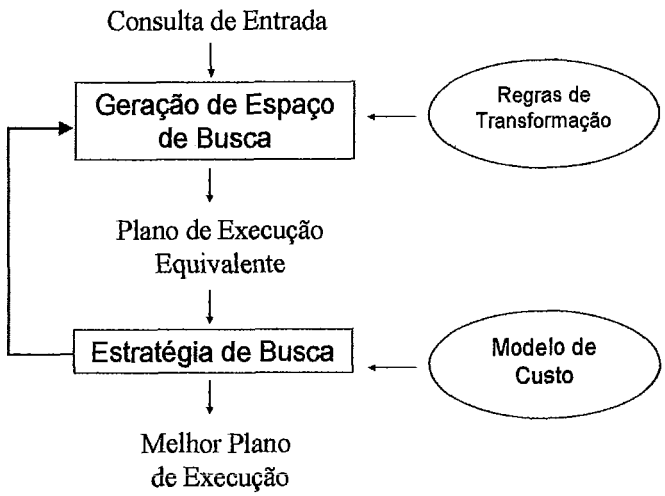


Figura 7 - Processo de otimização de consultas (VALDURIEZ e ÖZSU, 2001)

A otimização da materialização de documentos AXML, proposta pelo XCraft, acontece de forma análoga ao processo da Figura 3. Obviamente existem detalhes inerentes ao problema, os quais o tornam diferente. A Figura 8 apresenta o processo de materialização de documentos AXML de forma detalhada, segundo os conceitos enunciados acima.

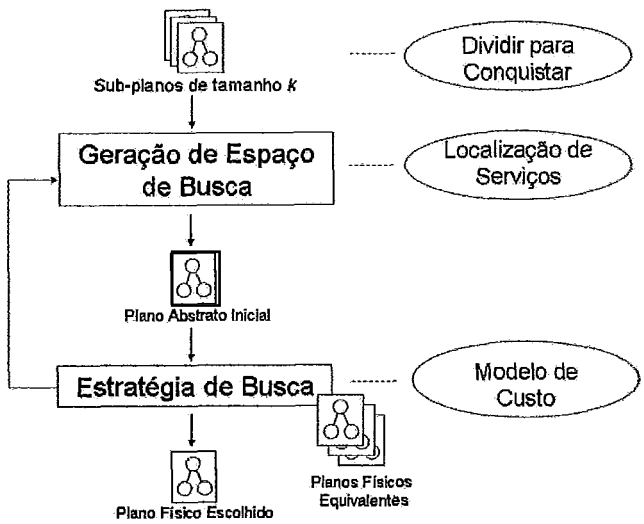


Figura 8 - Otimização de Documentos AXML

A partir do grafo de dependências de um documento AXML, são extraídas suas árvores de expansão pela aplicação da heurística dividir para conquistar. As árvores de

expansão, que são analisadas independentemente, são quebradas em árvores ainda menores de altura *k*. Este procedimento visa diminuir a complexidade do plano e acelerar a obtenção resultados. Conforme as informações coletadas pelo localizador de serviços, cada uma das chamadas de serviço de cada árvore é anotada com os possíveis executores e delegados, gerando o plano abstrato inicial, que representa o espaço de busca do plano a ser analisado pelo componente estratégia de busca. É justamente neste componente que diferentes métodos e estratégias de busca e otimização podem ser empregados e onde aplicamos a SLS-MC. Veremos a seguir, que a materialização de documentos AXML pode ser vista como um problema de otimização global, mas que devido a sua complexidade, precisa ser resolvido por meio de estratégias que fazem uso de meta-heurísticas ou de métodos que se adequam às características do problema e tentam ao máximo encontrar o mínimo global do espaço de busca.

3.1 – Problemas de Busca

Problemas de otimização global são geralmente problemas combinatórios que tem por objetivo encontrar a melhor solução de todas, ou seja, o mínimo global de um espaço de busca. A idéia não é encontrar um mínimo local, mas o menor mínimo local de todo o espaço de busca.

Problemas de otimização global podem ser definidos da seguinte maneira: dado um conjunto finito, fechado e não vazio $D \subset \mathcal{R}^n$ e uma função contínua de valores reais f:A $\rightarrow \mathcal{R}$, onde $A \subset \mathcal{R}^n$ é um conjunto adequado que contém D, encontre pelo menos um ponto $x^* \in D$ que satisfaça $f(x^*) \leq f(x)$ para todo $x \in D$ ou mostre que tal ponto não existe (HORST, HOANG et al., 1996).

Para encontrar o ótimo global de um problema combinatório, ou **de problemas de busca**, como são comumente chamados, em função de suas características, (COLORNI, DORIGO et al., 2006) é teoricamente possível enumerar todas as possibilidade e avaliá-las. Acontece que muitos destes problemas são tipicamente difíceis de serem resolvidos porque possuem espaços de busca gigantescos que levariam muito tempo para serem percorridos. A estratégia de percorrer todo o espaço de busca acaba sendo, portanto tanto indesejável quanto não rastreável em função do crescimento exponencial de alguns espaços de solução (SCRIPTOR, 2006).

Consequentemente, muitas estratégias foram desenvolvidas com objetivo de encontrar soluções parcialmente ótimas em espaços de busca muito grandes, sem a

necessidade de avaliar todas as possibilidades. Estas estratégias adotam algoritmos de aproximação que sacrificam a garantia de encontrar mínimos globais em troca da obtenção de boas soluções em tempos significantemente reduzidos. Em suma, estas estratégias procuram resolver o seguinte questionamento: dado um espaço de soluções *S*, uma função de custo *f* e um limite θ, existe uma solução viável $i \in S$ *tal que* $f(i) \leq \theta$?

Nos últimos 20 anos, surgiu um novo tipo de algoritmo de aproximação que tenta basicamente combinar métodos básicos de heurísticas com *frameworks* de alto nível com o objetivo de explorar eficientemente o espaço de busca. Estes métodos são comumente chamados de **meta-heurísticas** (BLUM e ROLI, 2003). Meta-heurísticas são tipicamente estratégias de alto nível que guiam uma heurística específica de um problema base com o objetivo de aumentar seu desempenho (STUTZLE, 1998a). Geralmente, as meta-heuristicas não são específicas a um problema e possuem mecanismos para evitar ficarem presas em mínimos locais. Algumas meta-heuristicas evoluíram bastante principalmente em função de problemas NP - completos modernos que precisam da ajuda de heurísticas para encontrar boas soluções (SCRIPTOR, 2006).

Em seguida, faremos um breve estudo sobre algumas meta-heuristicas e métodos mais conhecidos e aplicados em problemas de otimização global tais como busca gulosa, programação dinâmica, algoritmos de busca local, como subida de encosta, têmpera simulada e busca tabu, procedimento de busca adaptativas gulosas e Randômicas - PBAGR (*Greedy Randomized Adaptative Search Procedure* - GRASP) e ramificação e poda (*Branch & Bound*).

3.1.1 – Busca Gulosa

A busca gulosa tenta expandir o nó mais próximo à meta, na suposição de que isso provavelmente levará a uma solução rápida. Desse modo, ela avalia os nós usando apenas a função heurística: $f(n) = h(n)$ (RUSSEL e NORVIG, 2003). Ou seja, a busca gulosa sempre toma a decisão que lhe parece melhor no momento: ela toma uma decisão local ótima na esperança de que esta escolha a leve a uma solução global ótima (CORMEN, LEISERSON et al., 2001).

Os algoritmos de busca gulosa nem sempre encontram a solução ótima, mas funcionam bem para alguns tipos de problemas, principalmente por se tratar de uma estratégia bastante rápida já que o algoritmo não perde muito tempo analisando outras alternativas.

Na materialização de documentos AXML os planos precisam ser analisados inteiramente com o objetivo de aproveitar bem os *links* de comunicação e de perceber bons pontos de delegação. Desta maneira, ótimos locais não garantem ótimos globais e a estratégia gulosa não é indicada para a materialização de documentos AXML.

3.1.2 – Programação Dinâmica

A programação dinâmica (CORMEN, LEISERSON et al., 2001), tal como o método dividir para conquistar soluciona problemas através da combinação da solução de subproblemas. O método dividir para conquistar quebra um problema em subproblemas independentes, os soluciona recursivamente e depois combina suas soluções para resolver o problema original. Ao contrário, a programação dinâmica é aplicável quando os sub-problemas não são independentes, isto é, quando sub-problemas compartilham sub-sub-problemas. Um algoritmo de programação dinâmica resolve cada sub-problema só uma vez e depois salva suas respostas em uma tabela, evitando de ter que resolve-lo novamente cada vez que é encontrado.

Assim, existem dois ingredientes chaves que um problema de otimização deve possuir para que fique caracterizada a aplicabilidade de programação dinâmica: sub-estrutura ótima e sobreposição de problemas. Um problema possui sub-estrutura ótima, se sua solução ótima é composta pelas soluções ótimas de seus sub-problemas. Por problemas sobrepostos queremos dizer o espaço de subproblemas deve ser pequeno de tal modo que o algoritmo recursivo possa resolver os mesmos problemas várias vezes, ao invés de estar sempre gerando novos sub-problemas (CORMEN, LEISERSON et al., 2001).

A materialização de documentos AXML não atende a estas duas características. Uma solução ótima para um sub-plano não garante a solução ótima do plano global. Isso se deve a possibilidade de delegação de chamadas, que pode permitir uma redução de custo por fazer uso de *links* mais rápidos. Os planos alternativos precisam ser comparados inteiramente de tal modo que seja possível "perceber" bons pontos de delegação. Pelo mesmo motivo, mesmo para sub-planos pequenos, o espaço de busca pode ser muito grande e estes não se repetem com uma freqüência que justifique o armazenamento de resultados para posterior reutilização.

3.1.3 – Algoritmos de Busca Local

Algoritmos de busca local são tradicionalmente utilizados em problemas de otimização onde o caminho percorrido para se atingir o objetivo, ou seja, os vários estados analisados não importam. Geralmente, o objetivo dos algoritmos de busca local é encontrar o melhor plano ou solução que minimize (ou maximize) a função de custo objetivo do problema. Estes algoritmos iniciam a busca a partir de uma solução inicial e tentam encontrar iterativamente uma solução melhor em uma vizinhança *N(i)* propriamente definida da solução atual *i*. Assim, a busca local trabalha com apenas um estado corrente e em geral move-se apenas para estados vizinhos deste estado. A escolha de uma estrutura apropriada de vizinhança é crucial para o desempenho de um algoritmo de busca local e deve sempre ser sempre feita de uma maneira específica para problema sendo tratado (STUTZLE, 1998b).

Estes algoritmos são amplamente utilizados em problemas complexos como escalonamento de jornadas de trabalho e otimização de redes de comunicação (RUSSEL e NORVIG, 2003). Subida de encosta (*Hill Climbing*), têmpera simulada (*Simulated Annealing*) e busca Tabu são exemplos de algoritmos de busca local.

3.1.3.1 – Subida de Encosta

O algoritmo subida de encosta (RUSSEL e NORVIG, 2003) caracteriza-se por mover-se sempre em direção a valores menores (ou maiores dependendo do problema), mas que nem sempre são os melhores, em função de irregularidades na topologia do espaço de busca. É possível que a solução conduza a mínimos (ou máximos) locais, que são soluções menores que a de seus estados vizinhos, mas que não são as menores do espaço de busca. É, portanto, um algoritmo incompleto. Uma alternativa seria utilizar estados gerados de forma aleatória. Esta alternativa caracteriza-se por ser completa, porém pode ser bastante ineficiente. A Figura 9 apresenta o pseudocódigo do algoritmo de busca de subida de encosta tal como descrito em (RUSSEL e NORVIG, 2003)

```
função SUBIDA-DE-ENCOSTA retorna um estado que é um máximo global
entradas: problema, um problema
variáveis locais: corrente, um nó
                  vizinho, um nó

corrente ← CRIAR-NÓ (ESTADO-INICIAL[problema])
repita
  vizinho ← um sucessor de corrente com valor mais alto
  se VALOR[vizinho] ≤ VALOR[corrente] então retornar ESTADO[corrente]
  corrente ← vizinho
```

Figura 9 - Algoritmo de busca de Subida de Encosta

3.1.3.2 – Busca Tabu

A busca tabu é uma variante do algoritmo subida de encosta, que procura transcender o problema dos mínimos locais através de uma estratégia que proíbe ou penaliza certos movimentos. O critério de classificar um movimento como proibido, ou seja, um Tabu, é evitar a ocorrência ciclos (GLOVER, 1986).

Ao contrário da subida de encosta, que mantém apenas o estado corrente na memória (o valor *f(i*) da melhor solução i* encontrada até o momento)*, a Busca Tabu mantém informações sobre o itinerário das últimas soluções visitadas, ou seja, ela mantém uma lista de tabus de k estados visitados anteriormente. Esta informação é usada para guiar o movimento de i para a próxima solução j que será escolhida na vizinhança $N(i)$. O papel da memória é justamente restringir a escolha a algum subconjunto de $N(i)$, proibindo movimentos para algumas soluções vizinhas já visitadas (HERTZ, TAILLARD et al., 2006; RUSSEL e NORVIG, 2003). Assim, o espaço de busca vai sendo explorado: movendo, a cada iteração, da solução i para a melhor solução em $N(i)$. A busca Tabu não permite movimentos aleatórios. Ela prossegue aceitando soluções novas e possivelmente soluções piores somente se for para evitar uma solução já investigada.

Existem alguns problemas ainda abertos com relação a Busca Tabu tais como: a determinação de um período de proibição adequado e a adoção de algoritmos com complexidade computacional mínima para utilização de memória (SCRIPTOR, 2006).

3.1.3.3 – Têmpera Simulada

O algoritmo têmpera simulada (KIRKPATRICK, GELATT et al., 1983; RUSSEL e NORVIG, 2003) é um algoritmo probabilístico de busca local estocástica

que procura combinar a subida de encosta com a geração aleatória de estados, de tal modo que seja possível combinar eficiência e completeza. Este algoritmo baseia-se no mesmo processo de resfriar gradualmente um metal líquido até ele congelar. Os metais assumem uma configuração de baixa energia perfeita quando resfriados lentamente a partir de altas temperaturas. Isso corresponde ao fato que se o agendamento diminuir o valor de T suficientemente devagar o algoritmo irá encontrar um ótimo global (RUSSEL e NORVIG, 2003).

Basicamente, a têmpera simulada permite que estados vizinhos com custos piores sejam aceitos segundo um fator de probabilidade adaptativo menor que 1. Estes movimentos "morro acima" são usados para escapar da possibilidade de ficar preso em um mínimo local e são realizados com uma probabilidade que decresce à medida que a busca progride.

O algoritmo começa gerando uma solução inicial, de forma randômica ou através de heurísticas, e inicializando o valor do parâmetro de temperatura T. Depois, a cada iteração, uma nova solução $s \in N(i)$ (onde i corresponde a solução atual) é gerada randomicamente e aceita como a nova solução dependendo dos valores de $f(s)$, $f(i)$ e T. s substitui i se $f(s) < f(i)$ ou se $f(s) \geq f(i)$ com uma probabilidade que é uma função de T, e $f(s) - f(i)$. A probabilidade é geralmente calculada segundo a distribuição de Boltzmann (Eq 5) (BLUM e ROLI, 2003; SCRIPTOR, 2006).

$$e^{\left(-\frac{\Delta E}{T}\right)}, \qquad \textit{Eq 5}$$

onde ΔE é igual a $f(s) - f(i)$

A temperatura T vai decrescendo durante o processo de busca. Assim, no início da busca a probabilidade de aceitar movimentos "morro acima" é alta e vai decrescendo gradualmente, convergindo para um simples algoritmo iterativo e incremental (BLUM e ROLI, 2003).

O algoritmo da têmpera simulada pode ser bastante eficiente. Ele envolve um acordo entre o desejo de se obter uma solução de boa qualidade e a restrição de tempo computacional. A chance de obter uma solução ótima pode ser aumentada diminuindo-se o resfriamento, mas ao fazer isso, o algoritmo irá requerer mais tempo. Para usar adequadamente a Tempera Simulada, é preciso encontrar um "resfriamento" adequado que seja capaz de encontrar uma solução suficientemente boa sem, porém, sem gastar

muito tempo (SCRIPTOR, 2006). A Figura 10 apresenta o pseudocódigo da têmpera simulada como definido em (RUSSEL e NORVIG, 2003)

função TÊMPERA-SIMULADA(*problema, escalonamento*) **retorna** um estado solução

entradas: *problema*, um problema
escalonamento, um mapeamento de tempo para "temperatura"
variáveis locais: *corrente*, um nó
próximo, um nó
T, uma "temperatura" que controla a probabilidade de passos descendentes

corrente ← CRIAR-NÓ (ESTADO-INICIAL[*problema*])
para t ← 1 **até** ∞ faça
T ← escalonamento[t]
se *T* = 0 **então retornar** *corrente*
proximo ← um sucessor de *corrente* selecionado ao acaso
ΔE ← VALOR[*próximo*] – VALOR[*corrente*]
se $\Delta E > 0$ então *corrente* ← *próximo*
senão *corrente* ← *proximo* somente com probabilidade $e^{\Delta E/T}$

Figura 10 - Algoritmo de busca da Têmpera Simulada

3.1.4 – PBAGR *(GRASP)*

O Procedimento de Busca Adaptativa Gulosa e Randômica (PBAGR) (RESENDE e RIBEIRO, 2001) trata-se de um simples algoritmo iterativo de duas fases que combina heurísticas construtivas e busca local. A primeira fase do PBAGR consiste na construção da solução e a segunda na melhoria da mesma. A melhor solução de todas é retornada ao término do processo de busca.

O mecanismo de construção da primeira fase é constituído de dois ingredientes principais: uma heurística construtiva dinâmica e fatores aleatórios. Nessa fase, uma nova solução viável é construída adicionando-se elementos um a um à solução. A cada iteração, um novo elemento é escolhido aleatoriamente de uma lista de candidatos potenciais. A heurística é utilizada para pontuar e ordenar o valor de cada elemento, a cada passo sucessivo. A lista dos elementos candidatos, chamada de lista restrita de candidatos, é composta pelos melhores α elementos, aqueles elementos que ainda precisam ser inseridos na solução.

Os valores de cada elemento são sempre reavaliados em cada iteração e podem, portanto mudar de iteração para iteração. Por isso, esta heurística é dita dinâmica e faz com que o algoritmo seja adaptativo. A escolha randômica permite avaliar diferentes soluções a cada fase de construção. Aplicando-se inúmeras vezes a fase de construção, geram-se várias soluções iniciais a serem melhoradas pela busca local.

Em função da natureza aleatória da fase de construção, a solução gerada não tem chances de ser ótima, por isso a necessidade desta segunda etapa, que sucessivamente irá substituir a solução corrente por uma solução melhor, até que mais nenhuma outra seja encontrada na vizinhança. O algoritmo de busca local, usado na segunda fase, pode ser desde simples algoritmos como os iterativos e incrementais até algoritmos mais complexos, como têmpera simulada ou busca tabu (SCRIPTOR, 2006).

A principal idéia da estratégia PBAGR é gerar várias soluções inciais boas e melhora-las através da busca local, realizada em seguida à fase de construção. No entanto, a variabilidade das soluções iniciais está condicionada ao tamanho da lista de candidatos potenciais. Se a lista for pequena as soluções iniciais tendem a se repetir e a busca local não encontra soluções melhores que as já encontradas. Uma outra questão importante com relação à estratégia PBAGR é que existe uma independência entre as iterações e desta maneira não há um aprendizado a partir do histórico das soluções previamente encontradas (RESENDE e RIBEIRO, 2001). A Figura 11 apresenta o pseudo código do algoritmo PBAGR tal como definido em (RESENDE e RIBEIRO, 2001).

função GRASP (*num-max-iterações, semente*) retorna *melhor-solução*

entradas: *num-max-iterações*, um inteiro
semente, um inteiro que representa o tamanho da lista restrita de candidatos
variáveis locais: *solução*, um plano
melhor-solução, um plano

para *k* **de** 1 **até** Num-Max-Iterações **faça**
solução ← CONSTRUÇÃO GULOSA E RANDÔMICA (*semente*)
solução ← BUSCA-LOCAL(*solução*)
ATUALIZAR-SOLUÇÃO(*solução, melhor-solução*)
retornar *melhor-solução*

Figura 11 - Algoritmo de busca do PBAGR

3.1.5 – Algoritmos Genéticos

Algoritmos Genéticos (RUSSEL e NORVIG, 2003) são algoritmos de busca estocásticos, baseados em populações, inspirado na teoria da evolução de Darwin: estados sucessores são gerados pela combinação de dois estados pais, em vez de serem gerados pela modificação de um único estado.

Os algoritmos genéticos começam com um conjunto de k estados gerados aleatoriamente, chamado de população. Cada estado, ou indivíduo, é representado como uma cadeia sobre um alfabeto finito. Trata-se de uma analogia ao DNA que serve para codificar o problema e o espaço de busca. Esta codificação, no entanto pode ser bastante complexa e é preciso encontrar uma maneira de codificar o problema, de tal forma que seja possível realizar mutações, novas combinações e principalmente evoluir.

A produção da próxima geração de estados é realizada através da combinação de dois estados selecionados segundo uma pontuação de afinidade calculada através uma função de *fitness*. Para cada par a ser cruzado é escolhido ao acaso um ponto de *crossover* dentre as posições na cadeia. Assim os primeiros descendentes são gerados (RUSSEL e NORVIG, 2003).

Por fim, cada posição está sujeita à mutação aleatória com uma pequena probabilidade independente. A Figura 12 apresenta o pseudocódigo de busca usado pelos algoritmo genético segundo definido em (RUSSEL e NORVIG, 2003).

função ALGORITMO-GENÉTICO(*população*, FN-FITNESS) **retorna** um indivíduo

entradas: *população*, um conjunto de indivíduos
FN-FITNESS, uma função que mede a adaptação de um indivíduo

repita
  nova_população ← conjunto vazio
  para *i* ← 1 **até** TAMANHO(*população*) **faça**
    x ← SELEÇÃO-ALEATÓRIA(*população*, FN-FITNESS)
    Y ← SELEÇÃO-ALEATÓRIA(*população*, FN-FITNESS)
    filho ← REPRODUZ(x,y)
    se (pequena probabilidade aleatória) **então** *filho* ← MUTAÇÃO(*filho*)
    adicionar *filho* a *nova-população*
  população ← *nova-população*
até algum indivíduo estar adaptado o suficiente ou até ter decorrido tempo suficiente
retornar o melhor indivíduo em *população*, de acordo com FN-FITNESS

Figura 12 - Algoritmo Genético

Os algoritmos genéticos, no entanto, não são apropriados para toda os problemas de otimização possíveis. As soluções dos problemas precisam ser propriamente codificadas em cromossomos, o que nem sempre é possível. No caso da materialização de documentos AXML esta codificação e ainda mais difícil, pois é necessário que através dos cromossomos seja possível expressar a ordem de execução das chamadas e os sítios executores e delegados associados a cada tarefa. Além disso, é preciso respeitar o conjunto restrito de sítios para cada tarefa e a combinação entre genes não pode ser feita de forma aleatória.

3.1.6 – Ramificação e Poda (Branch & Bound)

Ramificação e poda é uma abordagem desenvolvida para solucionar problemas discretos e combinatórios de otimização. Como já mencionado, em problemas como estes não há condições computacionais de verificar se uma dada solução é ótima ou não. Como não há possibilidade de enumerar e comparar todas as alternativas, a técnica de ramificação e poda procurou fazer a comparação de forma implícita, o que resulta em uma enumeração parcial das alternativas (THE BRANCH AND BOUND APPROACH, 2006).

A base desta abordagem é uma árvore de enumeração total. No caso da materialização de documentos AXML, em cada nível nós alocamos uma chamada a um sítio. Nós consecutivos, representam agendamentos parciais e somente os nós folhas representam agendamentos completos. Assim, se em qualquer nó da árvore é possível mostrar que uma solução ótima não pode ocorrer em um de seus descendentes, não existe razão para considerar todos os nós descendentes. Ou seja, é possível podar a árvore naquele nó. Se for possível podar ramos suficientes, talvez seja possível reduzir a árvore em sub-árvores computacionalmente gerenciáveis. Para isso, é importante ter um bom valor inicial de referência. Quanto mais próximo este valor é da solução ótima, maiores são as oportunidades de poda.

A abordagem de ramificação e poda não é uma abordagem heurística, ou um procedimento de aproximação. Trata-se de um procedimento de otimização exato que encontra a solução ótima. Existe, porém uma limitação: esta abordagem não funciona quando a função de custo, usada para comparação das soluções alternativas, é **não monotônica**. Uma função *f(n)* é considerada monotônica se, para todo nó *n* e todo sucessor *n'* de *n* gerado por qualquer ação *a*, o custo estimado de alcançar o objetivo a partir de *n* não é maior que o custo do passo de se chegar a *n'* somado ao custo estimado de alcançar o objetivo a partir de *n'* (RUSSEL e NORVIG, 2003). Assim, não é possível garantir que a solução ótima não está em um dos ramos e é preciso averiguar todos. As funções de custo, usadas para comparar planos de materialização de documentos AXML são não monotônicas e por isso a técnica de ramificação e poda não pode ser aplicada.

A busca A* é um tipo de algoritmo de ramificação e poda. Ela avalia os nós combinando *g(n)*, o custo para alcançar cada nó e *h(n)*, o custo estimado do caminho mais econômico do no *n* até um nó objetivo e o custo para ir do nó *n* até o objetivo. A função de custo usada pela busca A* pode ser definida da seguinte maneira:

$$f(n) = g(n) + h(n)$$

Tendo em vista que *g(n)* fornece o custo de caminho desde o nó inicial até o nó *n* e que *h(n)* é o custo estimado do caminho de custo mais baixo desde *n* até o objetivo, temos que *f(n)* é igual ao custo estimado da solução de custo mais baixo passando por *n* (RUSSEL e NORVIG, 2003).

Os métodos descritos e analisados neste capítulo possuem diferentes características e devem ser aplicados segundo a conveniência. Para a materialização de documentos AXML, infelizmente nem todos os métodos são aplicáveis. A Tabela 2 traça um comparativo resumido entre os métodos apresentados, apresentando suas vantagens e desvantagens sob a ótica da materialização de documentos AXML. A estratégia exaustiva também é comparada.

Tabela 2 - Comparação entre os métodos de busca

Método/ Estratégia	Ótimo ou Aproximado	Vantagens	Desvantagens
Exaustivo	Ótimo	Garante a solução ótima	Proibitivamente lenta
Guloso	Aproximado	Rápido	Ótimos locais não garantem ótimos globais
Programação Dinâmica	Ótimo	Rápido	Ótimos locais não garantem ótimos globais
Subida de Encosta	Aproximado	Rápido Compara planos inteiros	Pode ficar presa em mínimos locais
Busca Tabu	Aproximado	Evita a análise de estados já analisados Compara planos inteiros	Uso intenso de memória
Têmpera Simulada	Aproximado	Grande probabilidade de encontrar a solução ótima Compara planos inteiros	Melhores resultados são obtidos mais lentamente
PBAGR	Aproximado	Permite gerar várias soluções iniciais potencialmente boas	Não há aprendizado a partir das soluções anteriormente geradas
Algoritmos Genéticos	Aproximado	Gera seletivamente boas soluções, eliminando naturalmente maus elementos	Dificuldade na representação dos indivíduos
Ramificação e Poda	Ótimo	Pode reduzir bastante a complexidade do problema através dos processos de ramificação e poda	Não se aplica aos problemas com funções de custo não monotônicas

Mediante o estudo realizado pudemos perceber que os algoritmos de busca local são os mais indicados para o problema da geração eficiente de planos de materialização de planos AXML, pois permite realizar a comparação de planos inteiros gerados através

de heurísticas diretamente ligadas ao problema. Estratégias que apresentam ainda medidas estocásticas, como a têmpera simulada, por exemplo, também constituem soluções interessantes porque permitem uma análise mais abrangente e eficiente do espaço de busca. Um característica interessante referente aos problemas de busca local refere-se a escolha do solução inicial. Acredita-se que através de uma boa solução inicial o algoritmo pode encontrar uma boa solução final utilizando menos passos para isso. A estratégia PBAGR apresenta uma alternativa para o problema através da geração de várias soluções iniciais potencialmente interessantes (quando a lista de candidatos potenciais não é muito restrita).

Desta maneira, inspirados pelas diferentes vantagens de cada uma destas estratégias e eliminando aquelas que não se aplicavam diretamente ao problema, nós propomos a SLS-MC, uma solução baseada em algoritmos de busca local estocásticos que utiliza heurísticas de agendamento de tarefas, tais como MET, min-min, max-min e duplex para gerar a solução inicial, utiliza heurísticas pertinentes ao problema para investigar apropriadamente a vizinhança e emprega ainda múltiplas condições de parada para determinar o fim da busca.

Capítulo 4 – SLS-MC – Busca Local Estocástica

Este capítulo apresenta a SLS-MC, uma solução baseada em algoritmos de busca local estocástica com múltiplas condições de parada para a otimização de documentos AXML. Além de descrever e detalhar seu funcionamento, este capítulo apresenta ainda os algoritmos utilizados em sua implementação.

A ausência de soluções prontas e apropriadas para o problema de materializar eficientemente documentos AXML nos incentivou a realizar um estudo sobre técnicas de busca e a desenvolver uma nova solução que levasse em conta todas as características até então não abordadas pelas soluções usadas na resolução de outros problemas semelhantes. Surgiu assim a **SLS-MC**, uma solução baseada em algoritmos de busca local estocásticos que combina múltiplas condições de parada e que tem por objetivo escolher bons planos de materialização para documentos AXML. Na verdade, ela procura evitar, sempre que possível, os piores planos e tenta encontrar dentro de um prazo admissível, um plano suficientemente bom, já que não é possível garantir que o melhor plano de todos será encontrado. Embora a estratégia SLS-MC tenha sido originalmente desenvolvida no contexto de documentos AXML, ela pode ser aplicada ao contexto de otimização de planos de execução de *workflows* baseados em serviços Web. A estratégia da SLS-MC foi inicialmente proposta em (PEREIRA, RUBERG et al., 2005) e os primeiros resultados, que comprovam a adequação da SLS-MC ao problema da materialização de documentos AXML, foram apresentados em (PEREIRA, RUBERG et al., 2006).

As seções a seguir apresentam a solução SLS-MC e descrevem seu funcionamento por meio dos algoritmos utilizados.

4.1 – Estratégia de Busca da SLS-MC

O objetivo da SLS-MC é claro: gerar eficientemente planos de materialização de documentos AXML e escolher dentro de um tempo aceitável de otimização, um bom plano de materialização. Isso significa que é preciso reduzir ainda mais os tempos de otimização obtidos pela estratégia iterativa e incremental do XCraft que é influenciada pelo tamanho dos espaços de busca dos sub-planos de altura k, que hoje são avaliados

de forma exaustiva. O número de planos equivalentes a serem avaliados aumenta consideravelmente à medida que o valor de *k* aumenta. O mesmo ocorre quando o valor de *k* não é muito alto, mas o número de chamadas, serviços e sítios referenciados pelos sub-planos implica inúmeras alternativas. Nosso objetivo é definir uma estratégia de busca que seja menos influenciada pelo tamanho de busca do problema e que seja, portanto mais rápida e eficiente.

Desta forma, optamos pela utilização de algoritmos de busca local estocásticos, como a têmpera simulada, que fazem a avaliação de planos inteiros e nos quais o tamanho do problema pouco influencia a eficiência da solução. Estes algoritmos permitem percorrer o espaço de busca de forma mais seletiva e inteligente.

Para melhorar ainda mais a eficiência da SLS-MC procuramos aplicar heurísticas de agendamento de tarefas para gerar a solução inicial e uma heurística de agrupamento pertinente ao problema para gerar os estados vizinhos e percorrer eficientemente o espaço de busca. Introduzimos ainda a aplicação de múltiplas condições de parada através das quais o usuário pode informar seus parâmetros de qualidade para o plano de materialização a ser obtido e permitir que o algoritmo pare tão logo uma boa solução (segunda suas definições) seja encontrada.

4.1.1 – Múltiplas condições de Parada

As soluções baseadas em busca local normalmente são executadas por um número fixo de iterações ou por um tempo determinado. Ao término da busca, a solução encontrada é retornada.

Esta abordagem é interessante, no sentido que permite que o espaço de busca seja devidamente varrido, mas pode ser ineficiente se uma boa solução for rapidamente encontrada. Ou seja, a busca sempre continua até satisfazer sua única condição de parada existente. Esta característica pode aumentar desnecessariamente o tempo de otimização.

Os algoritmos de busca local costumam ser eficientes em problemas de satisfabilidade proposicional, cujo objetivo é verificar se uma fórmula é satisfazível. Esta fórmula é composta por variáveis *booleanas* relacionadas por meio de operadores lógicos (FREUDER, NAREYEK et al., 2005; RUSSEL e NORVIG, 2003). A otimização de documentos AXML não pode ser vista como um problema de satisfabilidade, pois o problema não pode ser resolvido por meio de uma fórmula proposicional *booleana*. Através do uso de heurísticas e obedecendo as restrições do

plano, a busca local da SLS-MC sempre gera planos válidos que podem ser utilizados. No entanto, muitas vezes eles não são suficientemente bons e podem levar a longos tempos de otimização. É preciso, portanto criar uma formula através da qual seja possível verificar o momento certo de parar de procurar, seja porque uma boa solução já foi encontrada ou porque já foi gasto muito tempo tentando encontrá-la. É preciso também definir o que é uma boa solução de tal maneira a responder facilmente a questão: este plano é realmente bom? Será que não existem melhores?

Problemas de satisfação de restrições (RUSSEL e NORVIG, 2003) fazem uso de variáveis associadas a um domínio e a um conjunto de restrições. A satisfação de restrição é uma busca por variáveis que satisfaçam as restrições definidas. Quando há a necessidade de se otimizar o processo, basta acrescentar variáveis que definam a qualidade da solução (FREUDER, NAREYEK et al., 2005). A materialização de documentos AXML é composta de diversas restrições: as chamadas precisam ser executadas segundo a ordem do plano (primeiro os filhos e depois os pais) e os executores de cada chamada precisam ser escolhidos dentro de um conjunto determinado de sítios, que por sua vez, influenciam na escolha dos delegados.

A materialização eficiente de documentos AXML pode ser vista, portanto como um problema de satisfação de restrições, onde os algoritmos de busca local costumam ser empregados. Ao construir algoritmos baseados em heurísticas, como os de busca local, duas características principais precisam ser balanceadas: o grau de investigação e grau de exploração. O grau de investigação é a quantidade de esforço direcionada a busca local na região presente do espaço de busca (se uma região é promissora, é importante vasculha-la mais). Já o grau de exploração é a quantidade de esforço gasta na busca em regiões distintas do espaço de busca (muitas vezes, escolher uma solução em uma região distante ou aceitar uma pior, pode permitir a descoberta de soluções melhores).

Estas duas possibilidades são conflitantes e um bom acordo entre elas é muito importante e precisa ser ajustado. Outro acordo que precisa ser considerado é aquele entre o esforço (*i.e.* número de iterações) e eficácia (*i.e.* valor da solução final): de algum modo, o projeto de um bom método heurístico é um problema multi-atributo com dois objetivos conflitantes: esforço e eficiência (COLORNI, DORIGO et al., 2006).

Na SLS-MC, procuramos obter este acordo. Definimos várias variáveis de controle de tempo e qualidade que determinam as condições de parada da busca por bons planos de materialização. Por meio do emprego de várias condições de parada à

SLS-MC pretendemos diminuir ao máximo o tempo de otimização de documentos AXML sem, no entanto, influenciar a qualidade do plano escolhido. A busca na SLS-MC segue sempre até que qualquer uma das condições de parada seja alcançada. As condições de parada definidas foram as seguintes:

- **Ganho de custo** – Esta condição de parada é calculada em função do custo da solução inicial, ou seja, corresponde a uma porcentagem do custo da solução inicial que o usuário quer que a solução encontre independente do tempo que isso possa levar.

- **Ganho de tempo** – Esta condição de parada é calculada em função da complexidade do documento AXML (*i.e.* número máximo de planos alternativos de um documento). Corresponde a uma porcentagem da complexidade e determina o quanto do espaço de busca total o usuário deseja percorrer. Por esta condição, o usuário não se preocupa tanto com a qualidade do resultado, mas com o tempo (quanto mais planos mais tempo de otimização e, portanto uma chance maior de encontrar melhores soluções) que se leva para obter uma solução razoavelmente boa.

Estas duas condições podem gerar bons resultados rapidamente, atendendo principalmente aos requisitos do usuário que conhece, melhor do que ninguém, suas reais necessidades. No entanto, estas duas condições de parada, mesmo quando utilizadas em conjunto, podem não ser suficientes para limitar a busca, principalmente quando o espaço de busca é muito grande. Por exemplo: é possível que o ganho de custo determinado pelo usuário resulte em um custo inferior ao custo da solução ótima (que seria encontrado através de uma estratégia exaustiva). Esta solução nunca será encontrada. Se o documento AXML for muito complexo, mesmo com uma porcentagem pequena da complexidade (20%, por exemplo), o espaço de busca poderá ainda ser muito grande, levando dias e até mesmo anos para ser percorrido. Assim, a SLS-MC prevê condições de parada de segurança, que impedem que a busca siga por um tempo muito longo, além do tolerado pelo usuário. Estas condições representam tetos máximos impostos à otimização. As condições de segurança adotadas pela SLS-MC são:

- **Número máximo de planos** – Número máximo de planos que a SLS-MC pode avaliar.
- **Tempo máximo de otimização** – Tempo máximo que pode ser utilizado na otimização.
- **Número máximo de reinícios** – Número máximo de vezes que a busca pode recomeçar, ao se estabilizar.

A busca se estabiliza, quando os planos encontrados possuem custos muito próximos (dentro de uma margem pré-estabelecida em relação ao custo da solução atual). Ao atingir o número máximo de reinícios, a busca indica que encontrou uma solução boa e estável, difícil de ser melhorada. É importante frisar que por meio da definição destes parâmetros o usuário poderá definir explicitamente o que almeja, isto é, o que considera ser um bom resultado e quanto tempo está disposto a esperar por este bom resultado. Trata-se de uma abordagem mais evoluída que as empregadas atualmente que consideram apenas um número fixo de iterações ou a obtenção de um mínimo local.

4.1.2 – Funcionamento da SLS-MC

A estratégia de busca da SLS-MC pode ser descrita por meio de etapas como apresenta a Figura 13. Cada uma das etapas é descrita a seguir.

Na primeira etapa (1) a SLS-MC gera uma solução inicial ou um plano de materialização inicial a partir de um sub-plano abstrato inicial de altura k gerado pelo XCraft. Esta solução inicial é gerada de forma gulosa utilizando as anotações do sub-plano abstrato inicial e uma heurística de distribuição de tarefas. Existem várias heurísticas já adotadas em problemas de distribuição ou agendamento de tarefas em sistemas de processamento distribuído, tais como Grids computacionais e sistemas paralelos (BRAUN, SIEGEL et al., 2001; CASANOVA, LEGRAND et al., 2000). Na SLS-MC adaptamos quatro heurísticas ao problema da materialização de documentos AXML e procuramos avaliar quais destas contribuem mais na obtenção eficiente de bons planos de materialização. As quatro heurísticas consideradas foram: MET, Min-min, Max-min e Duplex. As definições sobre cada uma das quatro heurísticas empregadas nos itens a seguir foram extraídas de (BRAUN, SIEGEL et al., 2001).

- **MET** (*Minimum Execution Time*) – Esta heurística procura alocar uma tarefa (chamada) no processador (sítio) capaz de executá-la mais rapidamente. Pode parecer, impulsivamente, uma heurística boa e óbvia. Mas como esta não leva em conta as demais tarefas já alocadas, esta heurística pode sobrecarregar alguns sítios e nem sempre apresenta bons resultados quando existem muitas tarefas que podem executar paralelamente.
- **Min-min** - A heurística min-min se baseia na minimização do tempo de conclusão das tarefas, considerando para tal todas as tarefas ainda não mapeadas a processadores. Esta heurística assume um conjunto de tarefas não mapeadas a processadores e um conjunto de tempos mínimos de conclusão para cada uma destas tarefas (considerando-se todos os processadores). A tarefa com o menor tempo mínimo de conclusão é mapeada para o processador que permite este tempo mínimo. Esta tarefa é retirada do conjunto de tarefas não mapeadas e o processo se repete até que todas as tarefas sejam mapeadas.
- **Max-min** – A heurística max-min é semelhante à min-min. No entanto, a max-min leva em consideração os maiores tempos mínimos de conclusão e não os menores como a min-min.
- **Duplex** – A heurística duplex gera planos iniciais com as heurísticas min-min e max-min e depois escolhe aquele com menor tempo de execução.

As heurísticas MET, mim-min, max-min e duplex geralmente são utilizadas para o agendamento de tarefas independentes, que não é o nosso caso. Quando as tarefas são independentes qualquer tarefa pode ser escolhida para avaliação em qualquer momento. Em documentos AXML, no entanto, as chamadas possuem dependências de execução e não podem ser avaliadas em conjunto. Estas heurísticas foram então adaptadas e utilizadas seguindo a ordem topológica do plano, tal como em (BLYTHE, JAIN et al., 1998). Em cada iteração a SLS-MC considerada apenas as chamadas prontas, ou seja, as chamadas cujos filhos já tenham sido avaliados e agendados e que podem ser executadas de forma independente.

Na segunda etapa (2), a solução encontra o caminho crítico do plano gerado (caminho mais custoso desde o nó raiz até um nó folha) e realiza mudanças nas chamadas que fazem parte deste caminho. Estas mudanças seguem uma heurística de agrupamento que procura reduzir o tempo de execução do plano através da redução dos

custos de comunicação entre os sítios por meio da definição de pontos de delegação. Assim, a heurística escolhe para delegado de um nó, por exemplo, o executor de seu nó pai ou seu próprio executor. As mudanças são realizadas principalmente sobre os delegados das chamadas de serviço, uma vez que os executores já sofreram alguma otimização na definição da solução inicial. Ainda assim, em alguns casos, após algumas tentativas, ainda são realizadas escolhas aleatórias para os executores dos serviços com o objetivo de tentar encontrar melhoras pontuais no plano e investigar novos vizinhos.

Na etapa seguinte (3), após a realização das mudanças, o custo do plano é avaliado. Se houver diminuição no custo, as mudanças são acatadas. Caso contrário, segundo um **fator de probabilidade** definido pelo usuário, a SLS-MC aceita planos piores. O algoritmo têmpera simulada aceita planos piores segundo um fator de probabilidade *fp* menor que 1 definido em função da qualidade da solução atual e de um parâmetro de "resfriamento" *T*. De forma semelhante, a SLS-MC, também adotou um fator de probabilidade para a aceitação de planos com custos maiores. Quando $fp=0$ a solução toma sempre decisões gulosas, igualando-se ao algoritmo subida de encosta que se move sempre na direção de estados melhores. A configuração de valores baixos de probabilidade de aceitação pode ser bastante interessante quando o espaço de busca é pequeno e existe uma probabilidade maior de um mínimo local ser o mínimo global. Quando $fp=1$ a solução comporta-se como uma solução puramente aleatória, o que pode ser desejável, quando o espaço de busca é bastante grande. A introdução de uma maior aleatoriedade pode permitir a exploração de várias áreas distintas do espaço de busca e aumentar a possibilidade de se obter boas soluções. A escolha do valor de *fp* deve, portanto expressar a freqüência com a qual o usuário pretende aceitar planos piores e deve ser tomada de tal forma que ele seja adequado ao tamanho do problema, permitindo fugir dos mínimos locais e platôs, porém sem tornar a busca puramente aleatória. Concluindo, problemas muito complexos devem permitir uma maior probabilidade de aceitação enquanto que problemas menos complexos, devem permitir uma menor probabilidade ou até mesmo $fp=0$.

Uma alternativa interessante é fazer o valor de *fp* adaptativo como na têmpera simulada, onde a probabilidade diminui exponencialmente com a "má qualidade" do movimento. Movimentos piores têm maior probabilidade de serem aceitos no início e depois se tornam mais improváveis à medida que a solução começa a convergir para a solução final.

A aceitação de planos piores na SLS-MC ocorre quando a fórmula representada em Eq3 é verdadeira.

$$1 - fp < \textit{número aleatório gerado entre 0 e 1} \qquad \text{Eq3}$$

Na quarta etapa (4), a SLS-MC executa uma busca pelos melhores chamadores com o objetivo de permitir que não só os nós do caminho crítico sofram mudanças, mas que todos os chamadores do plano possam ser investigados também. Esta é mais uma medida de investigação de vizinhança da solução atual que permite que as tarefas penduradas em alguma tarefa do caminho crítico possam "aproveitar" o novo agendamento destas tarefas e contribuir para a diminuição do tempo de execução do plano. As mudanças são acatadas somente se houver uma diminuição de custo em relação ao melhor custo obtido até o momento.

Em seguida (5), ela realiza uma análise de **estabilização** da busca. A partir de um fator de estabilização definido pelo usuário, a SLS-MC calcula uma margem de custo a partir da solução atual dentro da qual o próximo plano a ser investigado é considerado equivalente ao plano atual. Desta maneira, se o custo de um plano fica próximo (de acordo com a margem calculada) do custo do plano atual, a contabilização de planos estabilizados é incrementada. Se o número máximo de planos estabilizados permitidos pela busca for atingido, a SLS-MC gera um novo plano totalmente aleatório (etapa 6): ocorre o que chamamos de **reinício**. Esta medida permite que a busca saia de platôs que são "espaços" onde os custos das alternativas avaliadas variam muito pouco. Se o número máximo de reinícios permitidos também for atingido, a SLS-MC interrompe a busca, indicando que uma boa solução já foi encontrada. Se algum plano divergir do custo dos demais planos, a contagem de planos estabilizados recomeça e o custo do plano atual é utilizado como referência.

As etapas de 2 a 5 são repetidas até que alguma das condições de parada seja alcançada (etapa 7). Durante toda a busca o melhor plano de todos é armazenado para que possa ser utilizado quando a busca terminar. Quando a busca termina o otimizador converte o plano encontrado pela SLS-MC em um plano físico e materializa o documento segundo as definições do plano físico obtido.

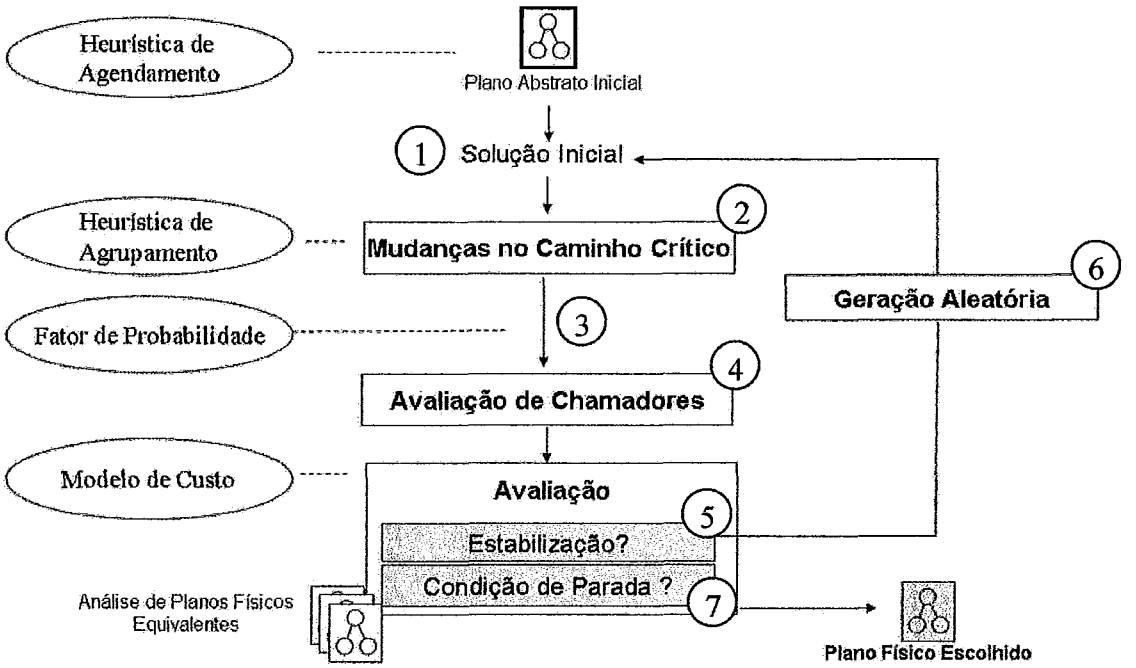


Figura 13 - Estratégia de busca da SLS-MC

4.2 – Algoritmos

Os algoritmos que se seguem apresentam a implementação das etapas da SLS-MC. O algoritmo da Figura 14 apresenta o código responsável por encontrar a solução inicial. Baseado na heurística utilizada, a função BUILD-INITIAL-SOLUTION chama o algoritmo responsável pela distribuição das chamadas pelos executores. Se a heurística escolhida for a MET, a função chama a função ALLOCATE-MINIMUM-EXECUTION-MACHINE, descrita no algoritmo da Figura 15. Caso contrário, se a heurística for a mim-min, max-min ou duplex, a função BUILD-INITIAL-SOLUTION executa a função FIND-MINIMUM-EXECUTION-TIME descrita na Figura 16.

```
função BUILD-INITIAL-SOLUTION(heurística, plano) retorna solução-inicial

    entradas: heurística, heurística a ser utilizada (MET, min-min, max-min ou duplex)
              plano, sub-plano abstrato incial de altura k gerado pelo XCraft

    variáveis globais: chamadas-não-mapeadas, lista de tarefas que ainda não foram
                       alocadas a um recurso
                       fila-de-execução, cada sitio executor possui uma fila própria de
                       tarefas a serem executadas

    variáveis locais: chamada-mapeada, uma chamada selecionada para ser mapeada no
                      recurso escolhido para ela
                      solução-inicial, plano modificado após a alocação de todas as tarefas

1   para cada filho da chamada raiz do plano faça
2     BUILD-INITIAL-SOLUTION(heuristica, filho)
3     se (heuristica <> "MET")
4       FIND-MINIMUM-EXECUTION-TIME(filho)
5     senao
6       se(heuristica = "MET")
7         ALLOCATE-MINIMUM-EXECUTION-MACHINE(filho);
8       fim se
9     fim se
10  fim para
11  se (heurisica <> "MET")
12    min ← infinito;
13    max ← 0;
14    enquanto houver chamadas-não-mapeadas faça
15        para cada chamada-não-mapeada em chamadas-não-mapeadas faça
16          se (heuristica = "minmin")
17            se (tempo-de-execução da chamada-não-mapeada < min)
18              chamada-mapeada ← chamada-não-mapeada
19            fim se
20          senao
21            se (tempo-de-execução da chamada não mapeada > max)
22              chamada-mapeada ← chamada não mapeada
23            fim se
24          fim se
25        fim para
26        adiciona chamada-mapeada na fila-de-execução do sítio executor de chamada-
    mapeada
27        enquanto houver chamadas-não-mapeadas faça
28          FIND-MINIMUM-EXECUTION-TIME (chamada-não-mapeada)
29        fim enquanto
30    fim enquanto
31  senao
32        ALLOCATE-MINIMUM-EXECUTION-MACHINE (chamada-não-mapeada)
33  fim se
34  retorna solução-inicial //plano alterado
```

Figura 14 - Algoritmo que constrói a solução inicial

A função ALLOCATE-MINIMUM-EXECUTION-MACHINE distribui as chamadas para aqueles executores que podem executá-las mais rapidamente. É utilizada pela heurística MET.

	função ALLOCATE-MINIMUM-EXECUTION-MACHINE(*chamada*) **retorna** *chamada*
	entradas: *chamada*, uma chamada a ser alocada ao recurso que permite executá-la mais rapidamente **constante**: *sítio-mestre*, um sítio que contém o documento a ser materializado
1	*menor-tempo-execucao* ← ∞
2	**para cada** *executor* dos possíveis executores de *chamada* **faça**
3	**se** (tempo de execucao da *chamada* no *executor* < *menor-tempo-execucao)*
4	executor da chamada ← *executor*
5	*menor-tempo-execucao* ← tempo de execucao da *chamada* no *executor*
6	**fim se**
7	**fim para**
8	chamador da chamada ← *sítio-mestre*
9	**retorna** *chamada* com executor e chamador definidos

Figura 15 - Algoritmo de distribuição da heurística MET

A função FIND-MINIMUM-EXECUTION-TIME analisa uma chamada e a adiciona à lista de chamadas ainda não alocadas (variável global). A análise escolhe o executor que consegue executar primeiramente a chamada, ou seja, aquele que apresenta o menor tempo de completude mínimo (*MCT – Minimium Completion Time)* (BLYTHE, JAIN et al., 1998; KWOK e AHMAD, 1999). A análise leva em consideração as chamadas já alocadas ao mesmo executor e contabiliza seus tempos para o cálculo do tempo de execução da chamada.

	função FIND-MINIMUM-EXECUTION-TIME *(chamada)* **retorna** *chamada*
	entradas: *chamada*, uma chamada a ser alocada ao recurso que permite concluí-la mais rapidamente **variáveis globais**: *chamadas-não-mapeadas*, lista de tarefas que ainda não foram alocadas a um recurso *fila-de-execução*, cada sitio executor possui uma fila própria de tarefas a serem executadas **constante**: *sítio-mestre*, um sítio que contém o documento a ser materializado
1	*menor-tempo-execucao* ← ∞
2	*tempo-execução* ← 0;
3	**para cada** *executor* dos possíveis executores de *chamada* **faça**
4	**para cada** *tarefa* da fila-de-execuçao do *executor* **faça**
5	*tempo-execução* += tempo de execução da *tarefa* no *executor*
6	**se** (*tempo-execução* < *menor-tempo-execucao)*
7	*menor-tempo-execucao* ← *tempo-execução*
8	*melhor-executor* ← *executor*
9	**fim se**
10	**fim para**
11	**fim para**
12	executor de *chamada* ← *melhor-executor*
13	chamador da chamada ← *sítio-mestre*
14	**adiciona** *chamada* a *chamadas-não-mapeadas*
15	**retorna** *chamada* com executor e chamador definidos

Figura 16 - Algoritmo de distribuição das heurísticas min-min, max-min e duplex

Para as heurísticas min-min e max-min, a função BUILD-INITIAL-SOLUTION da Figura 14, faz a distribuição das chamadas ainda não alocadas de acordo com as premissas de cada uma das heurísticas, ou seja, distribui primeiro as chamadas com a menor previsão de execução ou com a maior previsão de execução respectivamente. Para a heurística duplex, a função BUILD-INITIAL-SOLUTION é chamada para a heurística min-min e para a heurística max-min. O plano retornado com menor tempo de execuçao é escolhido. Para a heurística MET, o plano retornado pela função ALLOCATE-MINIMUM-EXECUTION-MACHINE já representa o plano inicial.

Após a definição do plano inicial, são realizadas modificações nos executores e/ou chamadores de alguns serviços Web com o objetivo de diminuir ainda mais o tempo de execução do plano. Estas modificações são realizadas sobre os nós que compõem o caminho crítico do plano.

O algoritmo da Figura 17 apresenta a função que encontra o caminho crítico do plano. O caminho crítico é determinado de forma gulosa calculando-se o custo (cumulativo) de cada nó a partir do nó raiz em direção aos nós folha.

```
função FIND-CRITICAL-PATH(chamada) retorna caminho-crítio
        entradas: chamada, uma chamada do caminho-critico a ter seus filhos investigados
        variável local: caminho-crítio, uma lista (ou sub-lista) de tarefas que compõem o
                        caminho mais custoso do plano
          comprimento-máximo ← 0
1         comprimento-chamada ← 0
2         para cada filho de chamada faça
3             comprimento-chamada ← custo do filho
4             se (comprimento-chamada > comprimento-máximo )
5               comprimento-máximo ← comprimento-chamada
6               chamada-máximo ← filho
7             fim se
8         fim para
9         adiciona chamada-máximo ao caminho-crítio
10        se(filho tem filhos)
11          FIND-CRITICAL-PATH(filho)
12        fim se
13      retorna caminho-crítio
14
```

Figura 17 - Algoritmo que encontra o caminho crítico

Finalmente, o algoritmo da Figura 18 apresenta as etapas da SLS-MC, encapsuladas na função DO-STOCHASTIC-SEARCH. Esta função demanda a construção da solução inicial, a definição do caminho crítico e a busca local estocástica.

função DO-STOCHASTIC-SEARCH(*heurística, plano*) **retorna** melhor plano encontrado

entradas: *heurística*, heurística a ser utilizada (MET, min-min, max-min ou duplex)
plano, sub-plano abstrato inicial de altura k gerado pelo XCraft

constantes: *fator-de-probabilidade*, decimal que determina a freqüência com que planos piores devem ser aceitos
fator-de-estabilização, decimal que determina a margem dentro da qual um plano é dito equivalente ao anterior
numero-máximo-de planos-estabilizados, inteiro que indica o número máximo de planos estabilizados permitidos para que ocorra um reinício
ganho-de-custo, decimal que indica o valor almejado para o *melhor-plano-geral*
ganho-de-tempo, decimal que indica o porcentagem do espaço de busca a ser percorrido
numero-máximo-restarts, inteiro que indica o número máximo de restarts permitidos
numero-máximo-planos, inteiro que indica o número máximo de planos a serem investigados
tempo-máximo, inteiro que indica o tempo máximo permitido para serem gastos na otimização (em minutos)

variáveis globais: *chamadas-não-mapeadas*, lista de tarefas que ainda não foram alocadas a um recurso
fila-de-execução, cada sitio executor possui uma fila própria de tarefas a serem executadas

variáveis locais: *próximo-plano*, plano sendo investigado
melhor-plano, plano aceito após modificações
melhor-plano-geral, melhor plano de todos os analisados encontrado até o momento

```
   //etapa 1
1  se (heuristica <> duplex)
2    solução-inicial ← BUILD-INITIAL-SOLUTION (heuristica, plano)
3  senao
4    solução-maxmin ← BUILD-INITIAL-SOLUTION ("maxmin", plano)
5    solução-minmin ← BUILD-INITIAL-SOLUTION ("minmin", plano)
6    se (custo solução-maxmin < custo solução-minmin)
7      solução-inicial ← solução-maxmin
8    senao
9      solução-inicial ← solução-minmin
10   fim se
11 fim se
12 melhor-plano ← solução-inicial
13 melhor-plano-geral ← solução-inicial
14 número-de-planos-estabilizados ← 0
15 custo-corrente ←0
16 restarts ← 0
17 contador-de-planos ← 0
   //etapa 7
18 enquanto (não alcançado ganho-de-custo e não alcançado ganho-de-tempo e contador-
19   de-planos < numero-máximo-planos e restarts < numero-máximo-restarts e não
20   esgotado tempo-máximo) faça

21   próximo-plano ← melhor-plano
22   caminho-critico ← FIND-CRITICAL-PATH (próximo-plano)
     //etapa 2
23   para cada chamada do caminho-critico faça
       // realiza alterações nos chamadores e executores de chamada
24     se chamador <> executor do pai
```

```
25              se puder delegar
26                  chamador ← executor do pai
27              fim se
28            senao
29              se chamador <> executor
30                  se  puder delegar
31                      chamador ← executor
32                  senão
                        executor ← chamador
33                  fim se
34              senao
35                  executor é escolhido randomicamente
36              fim se
37            fim se
38        fim para
39        se (próximo-plano < melhor-plano)
            //próximo plano com caminho crítico alterado
40          melhor-plano ← próximo-plano
41        senao
            //etapa 3
42          probabilidade ← número de 0 a 1 escolhido randomicamente
43          se (probabilidade > fator-de-probabilidade)
44              melhor-plano ← próximo-plano
45          fim se
46        fim se
          //etapa 4
47        AVALIAÇÃO-DOS-CHAMADORES
48        se (próximo-plano < melhor-plano)
              //próximo plano com chamadores já alterados
49            melhor-plano ← próximo-plano
50        fim se
51        se (melhor-plano< melhor-plano-geral)
52          melhor-plano-geral ← melhor-plano
53        fim se
54        custo-mínimo ← custo-corrente - (custo-corrente * fator-de-estabilização)
55        custo-máximo ← custo-corrente  + (custo-corrente * fator-de-estabilização)
          //etapa 5
56        se  (próximo-plano > custo-mínimo e próximo-plano < custo-máximo)
57          número-de-planos-estabilizados ++
58          se (número-de-planos-estabilizados = numero-máximo-de planos-estabilizados)
59              restarts ++
                //etapa 6 - reinício
60              próximo-plano ← GERAÇÃO-ALEATÓRIA
61            fim se
62        senao
63            custo-corrente ← custo de próximo-plano
64            número-de-planos-estabilizados ← 0
65        fim se
66        se (melhor-plano < melhor-plano-geral)
67            melhor-plano-geral ← melhor-plano
68        fim se
69        contador-de-planos ++
70    fim enquanto
71  retorna melhor-plano-geral
```

Figura 18 - Algoritmo da estratégia SLS-MC

Capítulo 5 – SiMAX – Simulador AXML

Este capítulo apresenta o SiMAX, um simulador para a plataforma Active XML desenvolvido com o objetivo de facilitar a realização de testes com diversas configurações e estratégias de busca para a materialização de documentos AXML. Através do SiMAX é possível realizar analisar o comportamento da SLS-MC e avaliar seus resultados comparando-os com outras estratégias e sob diversas configurações.

As aplicações executadas em ambientes distribuídos, como as de *workflows* e documentos AXML, têm em comum o fato de serem executadas em ambientes extremamente dinâmicos e heterogêneos, regidos por um grande número de variáveis que determinam a configuração e o comportamento destas aplicações. Todo este dinamismo introduz dificuldades na realização de testes que geralmente necessitam de um grande número de sítios distribuídos e conectados entre si, cada um com configurações próprias. Muitas vezes, esta configuração de sítios não é possível devido à falta de recursos ou infra-estrutura adequada. Outras vezes, é até possível, mas em função de políticas de segurança que restringem o acesso remoto aos sítios, acaba não sendo possível acessá-los e configurá-los adequadamente. Em virtude destas dificuldades e da inexistência de ambientes de simulação que atendam às nossas demandas para a realização de testes e às especificações do problema, tal como em (CASANOVA, LEGRAND et al., 2000) e (BRAUN, SIEGEL et al., 2001) nós também desenvolvemos nosso próprio simulador. O simulador, batizado de SiMAX, foi desenvolvido sobre a plataforma Active XML, com a qual pretendemos realizar os testes para analisar e validar a SLS-MC.

A proposta do SiMAX é permitir a configuração de uma rede P2P composta de sítios que fornecem serviços Web e armazenam documentos AXML, cujas chamadas de serviços demandam os servidos fornecidos na rede configurada. O SiMAX visa permitir também a otimização e a simulação da materialização dos documentos AXML armazenados nos repositórios dos sítios por meio da configuração de diferentes estratégias de otimização. Por essa razão, o SiMAX constitui-se em um ambiente simples e eficiente de testes no qual a configuração da infra-estrutura das redes P2P e a estratégia utilizada podem ser facilmente alteradas.

O SiMAX foi desenvolvido em JAVA e as configurações e informações necessárias para a simulação são passadas através de arquivos XML, convertidos em objetos através da biblioteca Commons Digester (JAKARTA COMMONS DIGESTER, 2006). O otimizador da plataforma Active XML, o XCraft, foi totalmente incorporado ao SiMAX e seus principais componentes foram adaptados de forma a obterem as informações a partir dos arquivos XML de configuração, ao invés do ambiente de execução. Desta maneira todo o processo de otimização é de fato executado pelo SiMAX, trazendo maior credibilidade aos resultados obtidos com a realização de testes.

A Figura 19 apresenta a arquitetura do simulador.

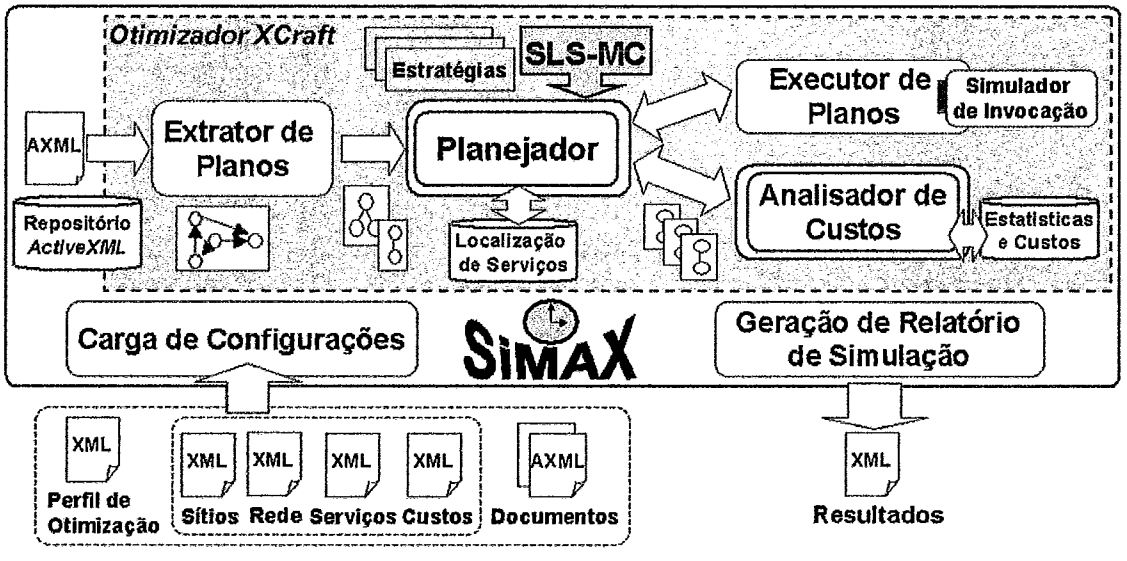


Figura 19 - Arquitetura do SiMAX

O SiMAX funciona da seguinte maneira: primeiro ele carrega as configurações necessárias para a simulação. Estas configurações estão contidas em vários arquivos XML e servem para popular catálogos de informação e estatísticas a serem utilizadas pelo simulador. Algumas destas configurações especificam o **perfil de otimização** do XCraft que consiste num conjunto de propriedades que determinam o comportamento do SiMAX na busca por bons planos de materialização. Por meio da realização de testes, procuramos descobrir qual o perfil de otimização que mais contribui para a obtenção de bons resultados, ou seja, que encontra bons planos de materialização em bons tempos de otimização. Um arquivo XML específico descreve o perfil de otimização. Ele contém, por exemplo, informações sobre a heurística a ser utilizada e valores das condições de parada, como o tempo máximo que o SiMAX deve gastar tentando obter um bom plano.

Além do arquivo que define o perfil de otimização do SiMAX, existem outros arquivos XML que contêm informações sobre os sítios do sistema P2P, os *links* de comunicação entre eles, os serviços Web disponíveis em cada sítio, os custos de execução dos serviços nos sítios e os documentos AXML que devem ser materializados.

Após a carga das configurações do ambiente de simulação, o SiMAX está pronto para disparar a análise da materialização de um documento AXML. Baseado nas configurações determinadas pelo usuário, o SiMAX pode materializar diretamente o documento ou otimizar a materialização segundo a estratégia de busca configurada. Quando o usuário escolhe otimizar a materialização, o documento AXML a ser materializado é encaminhado para o componente **Extrator de Planos** do XCraft, que extrai o grafo de dependências e gera um plano abstrato parcial. Esse plano é encaminhado ao componente **Planejador** do XCraft que utiliza as informações sobre localização dos serviços, carregadas a parir dos documentos XML, para anotar nos operadores do plano os respectivos sítios provedores e possíveis delegados, gerando um plano abstrato inicial. Conforme a estratégia de busca do perfil de otimização do SiMAX definida pelo usuário, o componente Planejador do XCraft usa o plano abstrato inicial para gerar e analisar planos físicos candidatos, solicitando ao componente **Analisador de Custos** para estimar o tempo de execução de cada plano gerado.

Com o objetivo de incorporar a SLS-MC na plataforma Active XML, o SiMAX estendeu o componente Planejador do XCraft, incorporando as heurísticas de agendamento de tarefas e acrescentando um componente que implementa todas as etapas da estratégia SLS-MC. As heurísticas implementadas foram: MET, min-min, max-min e duplex adaptadas para o contexto de tarefas dependentes.

Ao término da otimização, o melhor plano encontrado é encaminhado para o componente **Executor de Planos**. No Executor de Planos o SiMAX incorporou um mecanismo para a simulação da invocação de chamadas de serviços. A simulação de execução leva em conta os custos de delegação, o aumento no tempo total imposto pela manipulação de serviços Web, o atraso inerente do processador de cada sítio e os tempos de transferência das chamadas e de seus resultados.

No SiMAX existe um monitor que observa seu comportamento e produz relatórios contendo os resultados obtidos em uma simulação. Nestes relatórios são informados o tempo de otimização, o tempo de execução e o tempo total gasto na materialização de um documento AXML. São informados também o custo do plano

escolhido, o custo da solução inicial, a estratégia de busca, a heurística utilizada (quando é o caso), o número de planos avaliados e a condição de parada atingida.

Como já mencionado, apenas a execução dos planos de materialização escolhidos é simulada no SiMAX. Todo o processo de otimização é realmente executado, permitindo assim uma avaliação bem realista das estratégias simuladas.

5.1 – Configuração do Simulador

O SiMAX permite que uma série de informações seja definida pelo usuário. Por meio destas informações, configuradas através de arquivos XML e agrupadas em catálogos, o usuário pode definir quantos e quais sítios ele pretende utilizar. Pode determinar também a maneira como estes sítios estão interconectados e a capacidade de transmissão dos *links* que os interligam. Ainda é possível definir serviços Web e documentos AXML e determinar quais podem disponibilizá-los.

Os documentos AXML a serem materializados são representados de forma simplificada. Nesta representação são especificadas apenas as dependências de execução dos serviços Web demandados. As dependências são todas especificadas por meio de parâmetros concretos e não são previstos resultados intencionais, representados por novas chamadas a serem inseridas no documento. Ou seja, o documento e seu grafo de dependências não podem ser alterados em tempo de execução como ocorre de fato na plataforma AXML. Esta medida tem por objetivo facilitar a configuração de documentos AXML e simplificar a execução de testes. A Figura 20 mostra o exemplo de um documento AXML, tal como configurado no SiMAX e configurado em um arquivo chamado *d4.xml*. O documento AXML se chama "d4" e é composto de quatro chamadas aninhadas. A chamada "chamada4", por exemplo, é um parâmetro concreto da chamada "chamada3", e precisa, portanto ser executada primeiro que essa. O documento "d4" está armazenado no sítio *localhost*, ou seja, no sítio mestre, a partir do qual a materialização será demandada. Cada uma das chamadas demanda um serviço, cujo nome segue a mesma numeração do nome das chamadas. No exemplo da Figura 20 a chamada "chamada1" demanda o serviço configurado "serviço1", a chamada 2 o serviço "serviço2" e assim por diante. A definição de cada uma das chamadas de serviço, presentes nos documentos AXML, também é feita em um arquivo XML de configuração chamado *calls.xml*. Este arquivo contém além do nome das chamadas, os serviços demandados por essas e o tamanho dos resultados esperados. Desta forma, as

chamadas podem ser utilizadas mais de uma vez em um mesmo documento ou em documentos distintos.

```
<axmldocument name="d4">
  <axmlheader>
    <masterPeerURL>http://localhost:8080</masterPeerURL>
  </axmlheader>
  <sc id="Chamada1">
    <isFirstLevel>true</isFirstLevel>
  <sc id="Chamada2">
     <sc id="Chamada3">
        <sc id="Chamada4"/>
     </sc>
    </sc>
  </sc>
</axmldocument>
```

Figura 20 - Documento AXML do SiMAX

É possível configurar vários serviços Web e distribuí-los como necessário na topologia de rede criada no SiMAX. Um mesmo serviço Web pode ser executado em mais de um sítio e é possível configurar tempos diferentes de execução para cada par serviço/sítio. Os serviços também são definidos de forma simplificada em um arquivo XML denominado *services.xml*. Neste arquivo são definidos apenas os nomes dos serviços. No documento *serviceLocator.xml* e feita a distribuição dos serviços pelos sítios configurados e no arquivo *network.xml* são definidos as taxas de transferência entre os sítios da rede P2P, definidos através do arquivo *peers.xml*.

Além destas informações, é possível configurar também o perfil de otimização e os dados estatísticos da simulação. O perfil de otimização, determinado pelo arquivo *strategy.xml*, informa a estratégia a ser utilizada pelo otimizador, bem como todos os parâmetros necessários para a execução da estratégia escolhida. Os dados estatísticos, definidos no arquivo *statistics.xml*, são utilizados para informar os custos da realização das principais operações relativas a manipulação de serviços Web em cada um dos sítios.

Para entender melhor estas duas últimas categorias de propriedades e como elas influenciam o comportamento do simulador definiremos gramáticas que apresentam este conjunto de propriedades e seus possíveis valores.

5.1.1 – Perfil de Otimização

As propriedades do perfil de otimização determinam a estratégia e as heurísticas segundo as quais o SiMAX materializará os documentos AXML. A Tabela 3 apresenta

cada uma das propriedades que compõem o perfil de otimização e os valores que cada uma pode assumir.

Tabela 3 – Propriedades do perfil de otimização

Parâmetro	Descrição
Optimize	Indica se o simulador deve otimizar o documento a ser materializado.
Delegate	Caso haja otimização, determina se a delegação pode ser utilizada.
Materialize	Determina se o simulador deve materializar o documento.
Blocking	Indica se operadores de um plano podem ser executados paralelamente (uso de *threads*).
Split	Determina se planos devem ser quebrados em planos menores
SplitSize	Determina a altura k dos sub-planos
TopX	Indica os X primeiros planos a serem analisado em uma busca parcial
Search	Determina método de busca utilizado.
Heuristic	Indica a heurística de agendamento de tarefas a ser utilizada quando o método de busca for estocástico.
CostFactor	Percentual de ganho de custo relativo ao custo da solução inicial na condição de parada.
PlansFactor	Percentual de planos gerados relativo à complexidade do documento para parada.
PlansCeiling	Número máximo de planos que podem ser analisados.
HighCostProbability	Probabilidade de aceitação de planos piores para a solução estocástica.
StabilizationFactor	Percentual de custo que indica se uma solução estabilizou ou não.
NumPlansStabilize	Número de planos semelhantes para que a solução seja considerada estabilizada.
NumRestarts	Número máximo de reinícios permitidos.
TimeFactor	Determina o tempo máximo de otimização permitido

Segundo os dados da Tabela 3 é possível observar que um documento pode tanto ser materializado sem nenhuma otimização, como pode também ser otimizado sem que a materialização seja executada, isto é, o plano físico é gerado, porém não é materializado (ou neste caso, simulado).

A propriedade *Search* pode ser configurada como sendo *Exhaustive*, que determina que o SiMAX deve buscar exaustivamente o melhor plano, ou *Stochastic*, que determina que o SiMAX deve aplicar a SLS-MC. Para este último caso, as condições de paradas são determinadas pelas propriedades *CostFactor* e *PlansFactor*. A propriedade *PlansCeiling, TimeFactor e NumRestarts* representam as condições de parada de segurança que deverão ser obedecidas pela SLS-MC.

As propriedades *optimize*, *delegate*, *materialize*, *split*, *splitSize*, *topX* são propriedades originais do XCraft e que no SiMAX podem ser utilizadas em conjunto com a SLS-MC.

A Figura 21 apresenta ao lado de cada propriedade os possíveis valores que cada uma pode assumir. Os valores apresentados entre "*[]*" representam o conjunto das possibilidades possíveis. Quando estes valores são separados por "|", estes são mutuamente exclusivos e apenas um pode ser escolhido. Os valores delimitados por "*{}*" correspondem a valores compostos formados pelos caracteres apresentados no conjunto de valores possíveis. Por exemplo: a configuração {[0-9]} indica que a propriedade pode ser um número qualquer, formado pelos algarismos do intervalo de 0 a 9.

```
optimize = [true | false]
delegate = [true | false]
materialize = [true | false]
blocking = [true | false]
split = [true | false]
splitSize = { [0 -9 ] }
topX = { [0 -9 ] }
search = ["partiallyExhausted "| "Exhausted" | "Stochastic"]
heuristic = ["minmin" | "maxmin" | "MET" | "duplex"]
costFactor = [0-1].{ [0 -9 ] }
plansFactor = [0-1].{ [0 -9 ] }
plansCeiling = { [0 -9 ] }
highCostProbability = [0-1].{ [0 -9 ] }
stabilizationFactor = [0-1].{ [0 -9 ] }
numPlansStabilize = { [0 -9 ] }
numRestarts = { [0 -9 ] }
timeFactor = { [0 -9 ] }
```

Figura 21 - Perfil de otimização do SiMAX

5.1.2 – Propriedades Estatísticas e de Custos

As propriedades de estatísticas e de custos do SiMAX descrevem as características físicas do ambiente simulado. Elas são definidas para cada sítio envolvido e especificam os custos de comunicação de dados e de execução dos serviços Web. Essas propriedades são essenciais para as funções que estimam os custos dos planos de materialização. As principais propriedades de estatísticas e custos são apresentadas na Tabela 4.

Tabela 4 - Propriedades de estatísticas e de custos do SiMAX

Propriedade	Descrição
InitClientTime	Tempo necessário para inicializar os módulos de software no cliente.
InitServerTime	Tempo necessário para inicializar os módulos de software no cliente.
Pack	Tempo médio de empacotamento de 1 byte pelo SOAP.
Unpack	Tempo médio de desempacotamento de 1 byte pelo SOAP.
Parse	Tempo médio para compilar um fragmento AXML.
PlanAnalyseTime	Tempo médio gasto na análise de um plano de materialização.
Processing	Fator de carga de processamento de um sítio.

As propriedades estatísticas e de custos e seus possíveis valores podem ser analisados Figura 22.

```
initClientTime = [0-1].{ [0 -9 ] }
initServerTime = [0-1].{ [0 -9 ] }
pack = [0-1].{ [0 -9 ] }
unpack = [0-1].{ [0 -9 ] }
merge = [0-1].{ [0 -9 ] }
parse = [0-1].{ [0 -9 ] }
planAnalyseTime = [0-1].{ [0 -9 ] }
processing = [0-1].{ [0 -9 ] }
```

Figura 22 - Propriedades estatísticas e de custos

Este capítulo apresenta os resultados obtidos com os testes realizados com a SLS-MC no simulador SiMAX. Os testes têm por objetivo avaliar o comportamento e o desempenho da SLS-MC segundo as heurísticas e condições de parada configuradas no SiMAX. Sempre que possível os resultados foram comparados com os obtidos pelas estratégias exaustiva e gulosa.

6.1 – Configuração do SiMAX

Para testar o funcionamento e desempenho da solução SLS-MC procuramos realizar alguns testes comparativos no SiMAX usando a SLS-MC. Os testes procuram avaliar o desempenho da SLS-MC, analisando o comportamento com cada uma das heurísticas de agendamento incorporadas no SiMAX, e também com as diferentes condições de parada previstas na SLS-MC.

Sempre que possível os resultados obtidos com a SLS-MC são comparados com os resultados da estratégia exaustiva e de uma estratégia gulosa baseada na heurística MET. Essa última escolhe como executor de cada chamada o sítio capaz de executá-la mais rapidamente e define como delegado o sitio mestre. A comparação entre estas estratégias visa a averiguar o desempenho da SLS-MC em relação às estratégias utilizadas atualmente pela plataforma Active XML. Além disso, por meio da comparação com a solução exaustiva, podemos saber quanto foi possível melhorar o tempo de otimização em relação à busca pela solução ótima. No capítulo 2, diversas estratégias não exaustivas foram analisadas, entretanto suas limitações frente ao dinamismo das redes P2P não permitem abordar adequadamente a questão da delegação de chamadas de serviço. Assim, concentramos a comparação com a estratégia gulosa para observar se a SLS-MC através de sua busca local estocástica consegue, mesmo que levando mais tempo que a solução gulosa, melhorar a solução inicial gerada a tal ponto que o tempo total seja melhor que o obtido pela estratégia gulosa.

Os testes procuram averiguar ainda o impacto das heurísticas de agendamento e das condições de parada na obtenção dos resultados. Procuramos averiguar se as

heurísticas contribuem gerando soluções iniciais mais adequadas ao problema e se existe alguma heurística que produz resultados melhores e que é, portanto mais indicada a ser utilizada. Quanto às condições de parada, nós as analisamos separadamente. Se isoladas elas conseguirem obter bons resultados, certamente em conjunto os resultados poderão ser ainda melhores.

Para a realização dos testes, o SiMAX foi configurado com três documentos AXML, armazenados no sítio mestre e 15 serviços Web distribuídos entre 4 sítios heterogêneos de uma rede P2P, também configurada no SiMAX. Os documentos AXML, nomeados de d33, d65 e d129, possuem respectivamente 33, 65 e 129 chamadas, sendo uma delas a chamada raiz, ou seja, os documentos possuem uma chamada raiz e 32, 64 e 128 chamadas aninhadas com a altura fixada em h=3. Esta configuração permite que haja a comunicação de dados e a possibilidade de delegação de tarefas.

A chamada raiz de cada documento possui sempre três filhos que também são chamadas de serviço Web e têm fi=3, ou seja, cada chamada tem sempre três chamadas de serviço Web como parâmetros de entrada concretos (exceto as chamadas folhas). Esta medida permite que três tarefas independentes sejam sempre analisadas juntas por cada uma das heurísticas de agendamento.

Os quatro sítios que compõem a rede P2P usada nos testes são conectados através de *links* de comunicação com capacidades distintas. A topologia utilizada é apresentada na Figura 23.

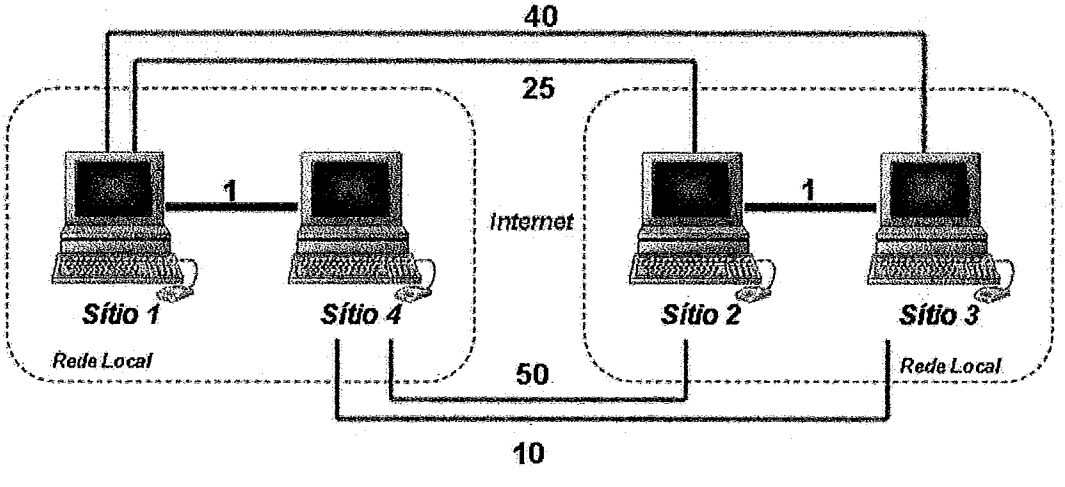


Figura 23 - Distribuição dos sítios

As linhas tracejadas agrupam os sítios em uma mesma rede local. Consequentemente, estes sítios possuem taxas de transferência mais altas. As linhas finas que ligam os sítios indicam comunicações roteadas pela Internet e que são mais lentas, por conseguinte. Os números sobre as conexões indicam o custo de comunicação entre os sítios. Esta informação é dada em relação aos *links* das redes locais. Por exemplo: o *link* entre os sítios 1 e 3 é 40 vezes mais lento que o *link* entre os sítios 1 e 4.

O sítio mestre é o Sítio 4. É ele que contém os documentos a serem materializados e para o qual os resultados, mesmo havendo delegações, precisam ser enviados para que os documentos possam ser devidamente materializados.

O número sobre cada sítio indica sua capacidade computacional, ou seja, quantas vezes cada sítio é mais lento em relação ao sítio 1. Por exemplo: o sítio mestre, é quatro vezes mais lento que o sítio 1. O sítio mestre, como um cliente de serviços, é em geral um sítio com capacidade de processamento mais modesta, enquanto que os sítios que provêem os serviços, em geral são máquinas dedicadas e com maior capacidade computacional.

A Tabela 5 apresenta o tempo de execução de cada um dos serviços Web em cada um dos quatro sítios configurados. Os valores (em milesegundos) foram calculados segundo uma distribuição uniforme, levando em conta a capacidade de processamento dos sítios. Assumimos, por simplicidade, que os resultados providos pelos serviços Web são sempre do mesmo tamanho para cada chamada, com valor fixado em 100 *Kbytes*.

Tabela 5 – Tempo de execução dos serviços em cada sítio

Serviço	Execução no Sítio 1	Execução no Sítio 2	Execução no Sítio 3	Execução no Sítio 4
1	*1*	*2*	*3*	*4*
2	*0,5*	*1*	*1,5*	*2*
3	*0,6*	*1,2*	*1,8*	*2,4*
4	*0,2*	*0,4*	*0,6*	*0,8*
5	*0,7*	*1,4*	*2,1*	*2,8*
6	*0,1*	*0,2*	*0,3*	*0,4*
7	*0,4*	*0,8*	*1,2*	*1,6*
8	*0,8*	*1,6*	*2,4*	*3,2*
9	*0,1*	*0,2*	*0,3*	*0,4*
10	*1*	*2*	*3*	*4*
11	*0,35*	*0,7*	*1,05*	*1,4*
12	*0,61*	*1,22*	*1,83*	*2,44*
13	*0,41*	*0,82*	*1,23*	*1,64*
14	*0,44*	*0,88*	*1,32*	*1,76*
15	*0,78*	*1,56*	*2,34*	*3,12*

Os valores relativos aos tempos de empacotamento e desempacotamento de mensagens, além de outras características ligadas aos serviços Web, documentos XML, e às propriedades estatísticas, foram configurados com os valores apresentados na Tabela 6.

Tabela 6 – Tempos relativos a dados XML e serviços Web

Sítio	InitClient Time	InitServer Time	Pack	Unpack	Merge	Parse	PlanAnalyse Time	Processing
1	*0,001*	*0,001*	*0,001*	*0,001*	*0,001*	*0,001*	*0,003*	*0,002*
2	*0,002*	*0,002*	*0,002*	*0,002*	*0,002*	*0,002*	*0,006*	*0,004*
3	*0,003*	*0,003*	*0,003*	*0,003*	*0,003*	*0,003*	*0,009*	*0,006*
4	*0,004*	*0,004*	*0,004*	*0,004*	*0,004*	*0,004*	*0,012*	*0,008*

Os valores atribuídos às propriedades da Tabela 6 também seguiram uma distribuição uniforme, obedecendo à capacidade de cada sítio.

Os valores atribuídos a cada uma das propriedades do perfil de otimização podem ser constatados na Tabela 7. Em alguns testes os valores sofreram variações segundo a condição de parada ou a heurística sendo avaliada.

Tabela 7 – Valores das propriedades de estratégia

Parâmetro	Valor(es) Atribuído(s)
Optimize	*true*
Delegate	*true*
Materialize	*true*
Blocking	*true*
Split	*false*
SplitSize	*1000*
TopX	*10*
Search	*exhaustive / stochastic*
Heuristic	*duplex / minmin / maxmin / MET*
CostFactor	*0,3*
PlansFactor	*0,05*
PlansCeiling	*500.000*
HighCostProbability	*0,5*
StabilizationFactor	*0,1*
NumPlansStabilize	*50*
NumRestarts	*20*
TimeFactor	*20*

Nos testes realizados consideramos um cenário onde todos os sítios podem executar todos os serviços Web e todas as chamadas de serviço podem ser delegadas a qualquer um dos quatro sítios considerados como executores. As configurações escolhidas para a realização dos testes ficam aquém das configurações de uma rede P2P real, onde o número de sítios é normalmente muito maior. No entanto, com apenas quatro sítios configurados em uma malha completa, 15 serviços Web distribuídos por estes quatro sítios e documentos com um grande número de chamadas (apesar de "baixos", os planos são bastante "largos") a complexidade dos documentos é enorme e permite avaliar de forma adequada a SLS-MC. O objetivo dos testes foi averiguar se a SLS-MC consegue encontrar bons planos de materialização em tempos aceitáveis de otimização independentemente da complexidade dos documentos e se é capaz de produzir tempos totais melhores que os das estratégias gulosa e exaustiva.

6.2 – Realização dos Testes

O planejamento a seguir foi usado nos testes para avaliar a estratégia SLS-MC, cada uma das heurísticas e as condições de parada:

- Foram utilizados três documentos: d33, d65 d129, cada um com respectivamente 33, 65 e 129 chamadas a serviços Web distribuídas em um plano com altura 3 e com *fun-in* 3. Para verificar a eficiência da SLS-MC em espaços de busca bem grandes, os planos originais não foram quebrados em sub-planos menores.
- Numa primeira bateria de testes, foram executados testes exaustivos com documentos menores, uma vez que com documentos muito complexos a aplicação de soluções exaustivas é inviável. Para a devida comparação, fizemos uma estimativa para documentos grandes com 33, 65 e 129 chamadas usando a estratégia exaustiva. Esta simulação considerou como tempo de análise de um plano 0.5 ms.
- Numa segunda bateria de testes, foi analisada a condição de parada **ganho de tempo** (condição de parada 1), calculado em 5% da complexidade dos documentos. Para planos muito complexos, como os analisados nos testes, mesmo 5% correspondem a um espaço de busca significativamente grande. Quanto maior a porcentagem da complexidade, mais demorada será a busca.
- Numa terceira bateria de testes, foi analisada a condição de parada **ganho de custo** (condição de parada 2), calculada em 50% do custo da solução inicial, gerada por cada uma das heurísticas. Se as soluções iniciais forem ruins acreditamos ser possível obter um plano com um custo inferior a 50% do custo da solução inicial rapidamente. No entanto, se a solução já for boa, a melhora pode demandar mais tempo. Quanto maior a porcentagem, provavelmente mais longa será a busca.
- Numa quarta bateria de testes, foram executados testes usando uma estratégia gulosa, baseada na heurística MET. Esta bateria de testes tem por objetivo verificar o potencial da SLS-MC em relação a estratégias gulosas normalmente utilizadas em problemas relacionados.
- Cada uma das baterias de testes foi realizada com as quatro heurísticas sendo analisadas, ou seja, MET, min-min, max-min e duplex, nesta ordem.

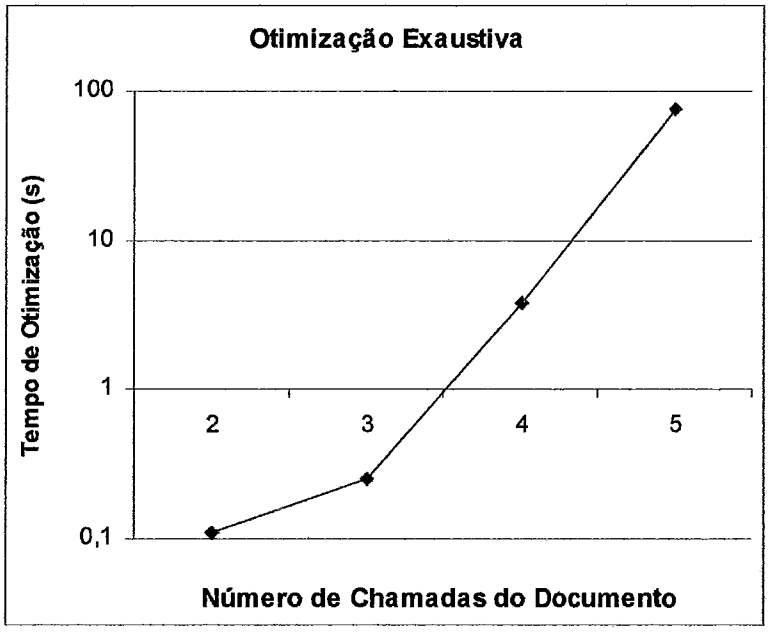
- Cada teste (condição de parada + heurística) foi realizado quatro vezes com cada um dos documentos configurados. Os resultados apresentados referem-se à média das quatro medições realizadas.
- O SiMAX foi instalado e configurado em um PC Centrino 1.73 Ghz, com 1 *Gbyte* de memória RAM e sistema operacional *Windows XP*.

6.3 – Avaliação dos Resultados

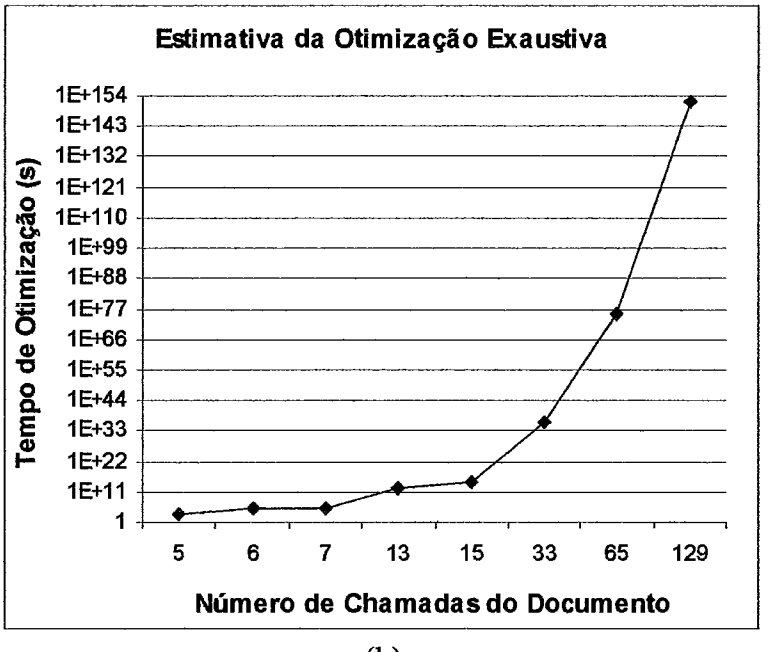
Os resultados apresentados referem-se aos testes executados segundo o planejamento apresentado na seção 6.2. Procuramos fazer uma avaliação sob diferentes pontos de vista, visando identificar claramente os detalhes da solução SLS-MC e das heurísticas utilizadas.

6.3.1 – Análise da Estratégia Exaustiva

Nesta bateria de testes, analisamos a estratégia exaustiva com o objetivo de determinar o tempo máximo de otimização dos planos. Todavia, verificou-se que a complexidade dos documentos avaliados é muito grande, tornando impossível a sua execução prática. Assim, geramos documentos pequenos, com no máximo 5 chamadas e medimos o tempo de otimização da solução exaustiva para esses documentos, cujos resultados estão na Figura 24(a). Para os documentos d33, d65 e d129, foi estimado o tempo de otimização a partir da respectiva complexidade, considerando-se um tempo médio de geração e avaliação de cada plano (estimado em 0,5 ms para o Sítio 1). Os resultados dessa simulação são mostrados na Figura 24(b).



(a)



(b)

Figura 24 - Tempo de otimização da estratégia exaustiva

Para documentos com apenas 7 chamadas o tempo de otimização já é de aproximadamente 37 horas. Para documentos com 33 chamadas a estratégia exaustiva torna-se impraticável. Ao compararmos os resultados simulados dos documentos d33, d65 e d129 com os resultados obtidos com a estratégia SLS-MC (seções 6.3.2 e 6.3.3), podemos concluir que só o tempo de otimização da estratégia exaustiva, ultrapassa em

muito, os tempos totais obtidos pela SLS-MC, independente da heurística utilizada. Mesmo a estratégia gulosa, baseada na heurística MET, apresentou resultados melhores.

Por ser uma abordagem mais escalável, a estratégia SLS-MC permite determinar o tempo de otimização conforme as condições de parada utilizadas, podendo-se fixar um limite dentro do qual seja possível encontrar um bom plano. Deste modo, a estratégia SLS-MC torna-se bastante interessante, pois permite que o desempenho do plano seja melhorado sem exigir a completa varredura do espaço de busca. Vale salientar que isso é bem adequado quando se trata de serviços Web. Na especificação do SLA (*Service Level Agreement*) (WEB SERVICES ACTIVITY, 2006) de um serviço Web, é comum o responsável pelo serviço determinar o tempo máximo de execução, sendo desejável que ele consiga também especificar o quanto deste tempo pode ser gasto em otimização.

6.3.2 – Análise da Qualidade do Plano Conforme a Heurística de Agendamento

A principal contribuição das heurísticas de distribuição de tarefas na SLS-MC, está na produção de um plano inicial que permita a SLS-MC evoluir rapidamente para um bom plano. Portanto, por meio desta bateria de testes pretendemos averiguar o impacto das heurísticas de agendamento de tarefas na qualidade dos planos obtidos pela estratégia de otimização e quanto a SLS-MC conseguiu melhorar os planos iniciais produzidos por cada heurística. Esta última análise nos permite averiguar o quão melhor a SLS-MC é em relação à estratégia gulosa, baseada em qualquer uma destas heurísticas.

Os resultados dessa bateria de testes são mostrados na Figura 25. Para esta bateria, a condição de parada foi fixada com o número de planos avaliados, calculado em 5% da complexidade (i.e., do número total de planos possíveis). Na Figura 25(a), é mostrado o custo do plano correspondente à solução inicial. A solução inicial é obtida conforme cada uma das heurísticas de agendamento de tarefas sendo analisada. Já na Figura 25(b) temos os custos finais dos planos obtidos após a otimização realizada pela SLS-MC tendo por base as soluções iniciais geradas pelas mesmas heurísticas.

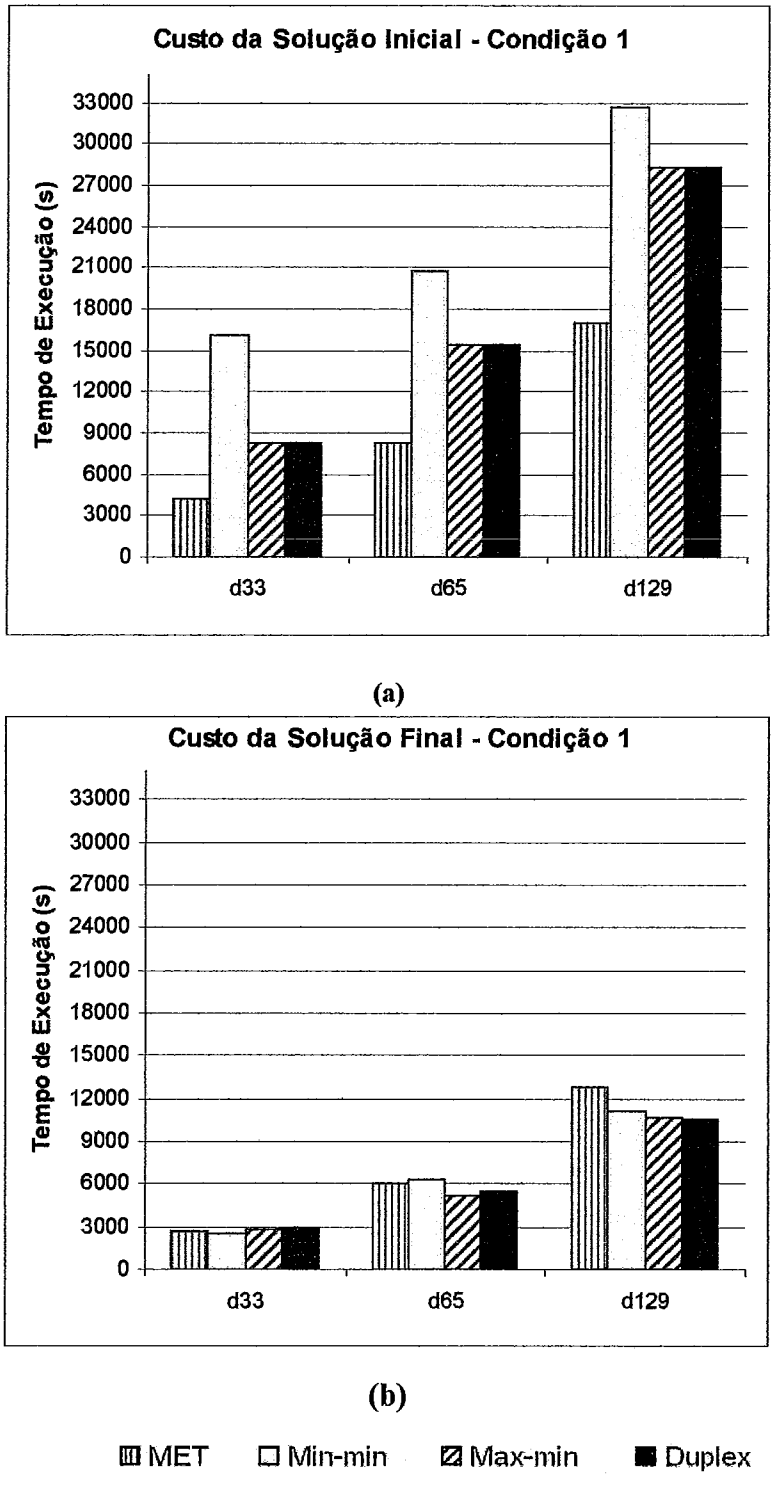


Figura 25 - Resultados de diferentes heurísticas de agendamento de tarefas

Comparando os custos iniciais e finais obtidos pela SLS-MC, podemos perceber que independente da heurística utilizada ela sempre conseguiu reduzir os custos das soluções iniciais.

Observando cada heurística isoladamente, verificamos que os planos finais ficaram com custos muito próximos, mesmo partindo de soluções ruins. Reparamos que a heurística min-min foi a heurística que produziu as piores soluções iniciais e mesmo assim os custos das soluções finais, obtidos pela mesma heurística, ficaram muito próximos das demais heurísticas que produziram soluções iniciais melhores. Aliás, a heurística MET foi a heurística que produziu as melhores soluções iniciais para todos os documentos. O custo da solução inicial da heurística MET chegou a ser 50% do valor da solução inicial gerada pela heurística min-min. Como o *fun-in* do grafo de dependências dos documentos não era muito grande *(fi=3)* o número de chamadas que podiam ser executadas em paralelo não era tão grande e a heurística MET acabou produzindo os melhores planos iniciais.

Apesar da superioridade da heurística MET nas soluções iniciais, esta, quando submetida ao mesmo tempo de otimização que as demais heurísticas, não contribuiu tanto nos tempos de execução. Sua contribuição decaiu à medida que número de chamadas aumentou.

Por meio do gráfico da Figura 26, podemos analisar o quanto a SLS-MC conseguiu melhorar cada uma das soluções iniciais geradas pelas heurísticas. Segundo os resultados, a SLS-MC conseguiu obter uma melhora de 84% em relação à solução inicial gerada pela heurística min-min para o documento *d33*! Trata-se de uma melhora bastante significativa. Se a otimização fosse baseada apenas nas heurísticas de agendamento, como usado normalmente (BRAUN, SIEGEL et al., 2001), teríamos soluções até 6 vezes pior . Em média, as melhoras ficaram em torno de 58%.

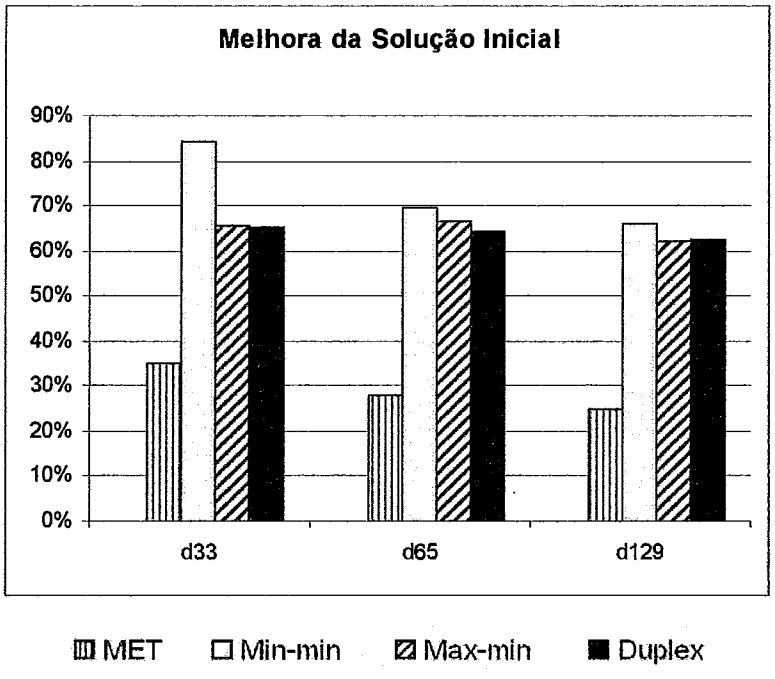


Figura 26 - Melhora da solução inicial

Estes resultados evidenciam o potencial da SLS-MC, principalmente em relação às soluções mais simples, baseadas simplesmente nas heurísticas de agendamento. A SLS-MC consegue produzir bons resultados mesmo quando as heurísticas de agendamento não fornecem boas soluções iniciais. De acordo com os resultados, ainda que partindo de soluções iniciais inferiores, a SLS-MC consegue obter soluções finais com custos muito próximos para todas as heurísticas com o mesmo tempo de otimização. Com a flexibilidade da SLS-MC o usuário poderá escolher a heurística que julgar ser mais adequada, sem, no entanto correr riscos de uma otimização mais demorada, caso a escolha não seja a mais adequada.

6.3.3 – Análise de Impacto da Condição de Parada

O objetivo desta bateria de testes é verificar o desempenho da SLS-MC conforme a condição de parada da estratégia. A idéia é analisar o tempo que cada heurística requer para alcançar resultados satisfatórios (ou estabilizar). As condições de parada avaliadas foram: número de planos avaliados (Condição 1), estabelecido em 5% da complexidade do documento; e ganho de custo (Condição 2), fixado em 50% do custo da solução inicial. Os testes foram repetidos com as quatro heurísticas e os três documentos sendo analisados.

Com a Condição 1 de parada (ganho de tempo) acreditamos ser possível obter boas soluções através da busca local estocástica analisando uma porção limitada do espaço de busca (5%). Já com a condição 2 de parada (ganho de custo), almejamos conseguir um plano com uma qualidade mínima, calculada em função do custo do plano inicial, independente do tempo que se leve para conseguir este plano. Esperamos com esta condição, que a busca não desperdice muito tempo tentando encontrar um bom plano, quando na verdade um plano suficientemente bom (determinado pelo usuário) já tenha sido encontrado. As heurísticas possuem um papel importante nesta avaliação. Acreditamos que podemos chegar mais rapidamente à solução almejada quando a solução inicial for boa. Se, no entanto, a solução inicial for realmente muito boa, a SLS-MC poderá não obter resultados com custos ainda menores e perder muito tempo tentando melhorar um resultado que já é na realidade muito bom. Este fator mostra claramente a importância da condição 1 de parada e a necessidade de utilizar múltiplas condições de parada. Ao atingir o número máximo de planos que o usuário pretende percorrer, a busca será interrompida. O usuário terá um plano com uma qualidade mínima dentro de um prazo aceitável, configurado por ele mesmo.

Obviamente, ao se atingir as condições de parada de segurança, como tempo máximo, número máximo de planos ou número máximo de reinícios, a busca também será interrompida.

Como a estratégia exaustiva é inviável para os documentos analisados, os resultados foram comparados com os resultados obtidos através da solução gulosa gerada a partir da heurística MET, tida como a mais simples e utilizada.

A percepção final do usuário sobre o ganho de tempo obtido com a SLS-MC se dá principalmente pela análise do tempo total, onde:

Tempo total = tempo de otimização + tempo de execução.

Pelo tempo total é possível constatar qual condição é mais vantajosa de um modo geral. Para alguns documentos e/ou heurísticas, acreditamos que vale mais a pena sacrificar a otimização em função da eficiência. Já para outros, é melhor sacrificar a eficiência em função da otimização (KWOK e AHMAD, 1999). Ou seja, em alguns casos, realizar uma rápida otimização, sem na verdade obter planos tão bons, pode sair mais barato (em termos de tempo) que gastar um bom tempo otimizando e obter um plano com tempo de execução menor. É através da análise do tempo total que

pretendemos identificar se a SLS-MC permite um bom acordo entre os tempos de execução e os tempos de otimização.

O gráfico da Figura 27 mostra os resultados obtidos com a condição de parada 1. Por ele é possível perceber que para cada um dos documentos, independente da heurística utilizada, os tempos totais ficaram abaixo dos custos totais obtidos pela estratégia gulosa, principalmente os documentos maiores.

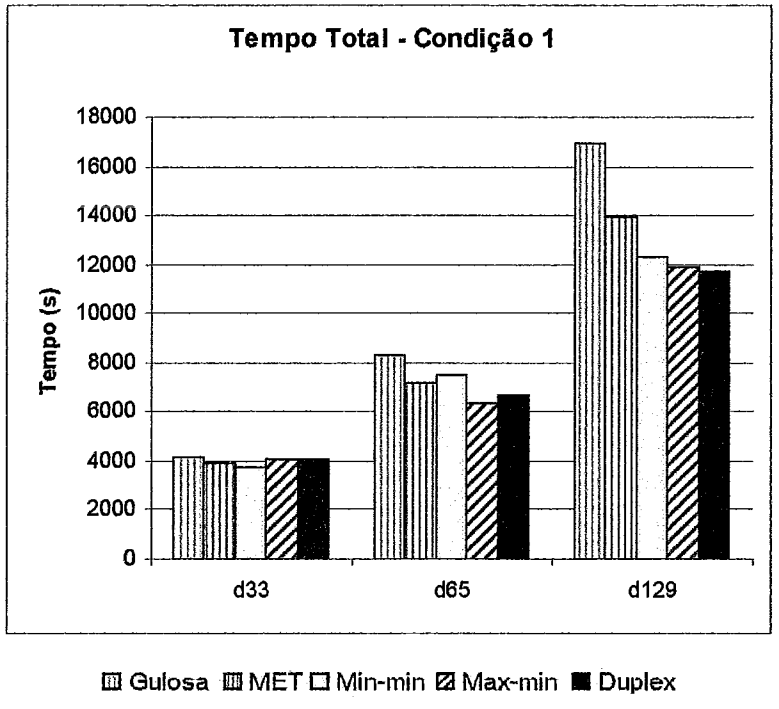


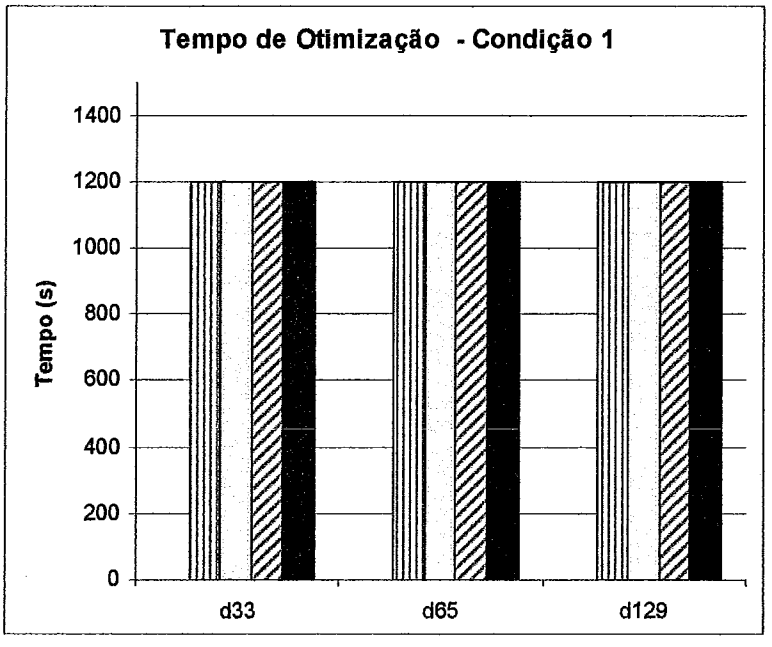
Figura 27 - Resultados da condição de parada 1

Analisando separadamente os fatores que compõem o tempo total, isto é, o tempo de otimização e o tempo de execução, podemos perceber qual deles tem maior influência sobre o tempo total e compreender melhor o comportamento da SLS-MC.

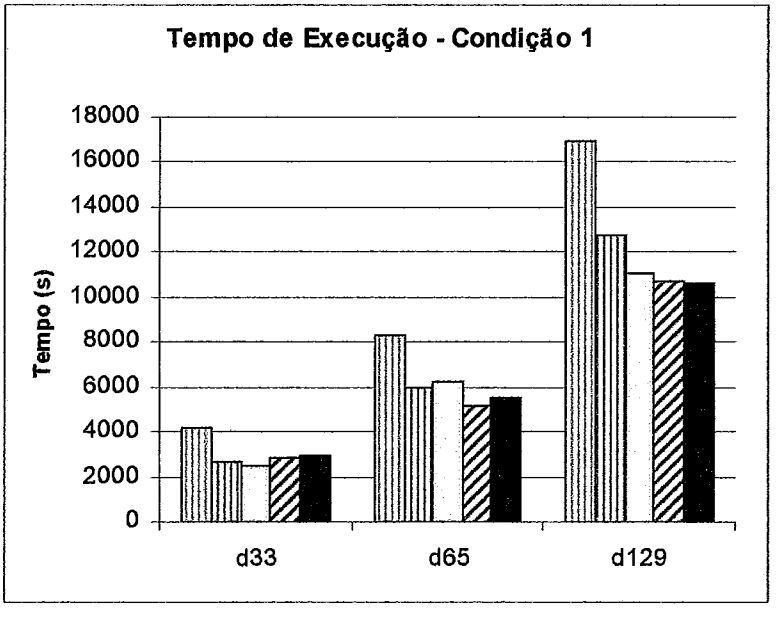
A Figura 28(a) apresenta os tempos de otimização obtidos através da condição de parada 1. A estratégia gulosa, por tomar decisões pontuais sem considerar planos inteiros, é extremamente rápida (e nem aparece gráfico!). Já a SLS-MC apresenta tempos de otimização mais altos. Isso se dá em função da abordagem baseada em busca local estocástica utilizada que tenta melhorar a solução inicial até que a condição de parada (5% da complexidade) seja atingida.

Nesta primeira condição de parada, como o tempo é dado em função da complexidade dos documentos, podemos constatar que, para os documentos analisados,

mesmo 5% da complexidade corresponde a um número muito grande de planos. A otimização acabou sendo concluída por ter atingido o tempo máximo de 20 minutos.



(a)



(b)

Gulosa MET Min-min Max-min Duplex

Figura 28 - Tempo de otimização e execução da condição de parada 1

Analisando os tempos de execução, Figura 28(b), podemos perceber que a "miopia" e rapidez da estratégia gulosa na fase de otimização, resultaram em tempos de execução muito altos e consequentemente em tempos totais mais altos que os obtidos

pela SLS-MC. Já o tempo gasto pela SLS-MC na otimização pode ser compensado na execução, pois os planos finais obtidos foram mais baratos, o que resultou em tempos totais mais baixos. Existe, portanto um bom acordo entre otimização e desempenho na SLS-MC.

O gráfico da Figura 29 apresenta os resultados obtidos com a condição de parada 2.

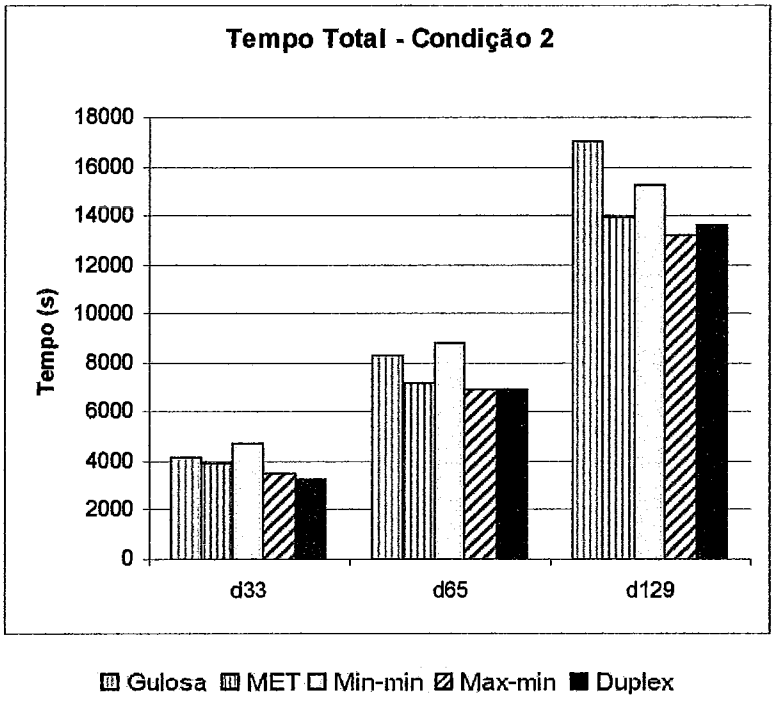
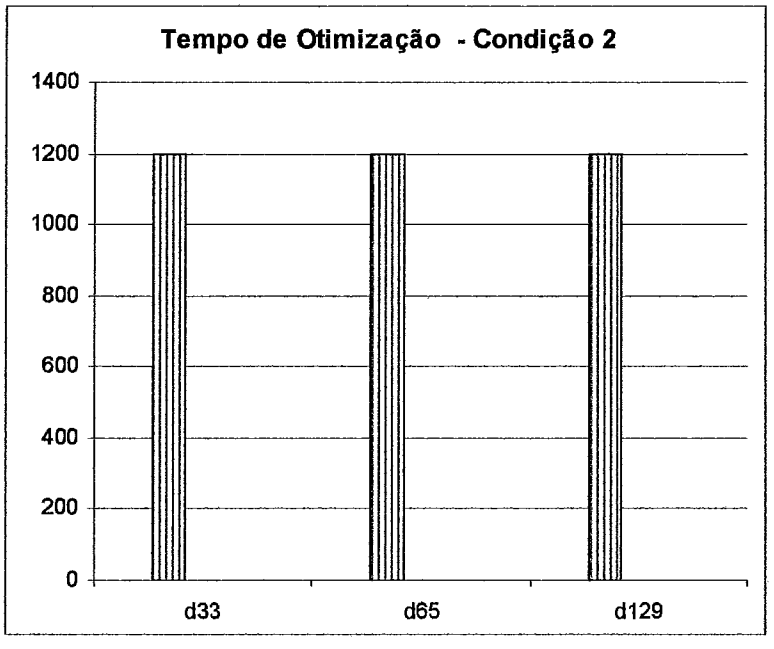


Figura 29 - Resultados da condição de parada 2

Segundo os resultados da Figura 29, quando a SLS-MC impõe uma restrição de qualidade, os resultados também ficam melhores que os da solução gulosa e para os documentos maiores, os tempos totais chegaram a ser inferiores aos obtidos com a condição de parada 1. Adicionalmente, podemos perceber uma pequena vantagem da heurística max-min sobre as demais e um desempenho ruim da heurística min-min. Segundo os resultados obtidos com a condição de parada 1, esta heurística, precisa de mais tempo para conseguir resultados melhores.

Analisando o gráfico da Figura 30(a), podemos perceber que a qualidade almejada (50% do custo inicial), pode ser rapidamente atingida (com valores muito menores que os obtidos pela heurística MET e por isso nem aparecem no gráfico), para as heurísticas min-min, max-min e duplex . Para a heurística MET, no entanto, percebemos que a otimização foi bem mais demorada. Isso se deve ao fato de a solução

inicial gerada por esta heurística, já ser muito boa e difícil de ser melhorada. Obter um plano, 50% mais barato que o plano inicial gerado pela heurística MET corresponderia a uma solução com um custo realmente muito baixo, inferiores aos obtidos por qualquer outra heurística nos 20 minutos da condição 1.



(a)

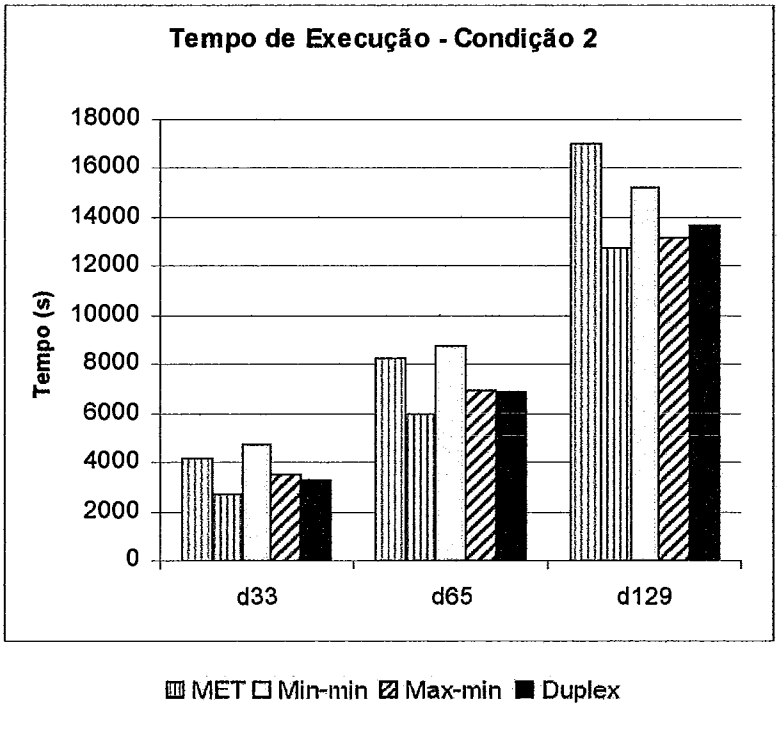


Figura 30 - Tempos de otimização e execução da condição de parada 2

Observando os tempos de execução, podemos concluir que mesmo com tempos tão pequenos de otimização, os custos dos planos finais almejados pelo usuário puderam ser atingidos. Para a maioria das heurísticas, os custos finais dos planos ficaram muito próximos aos custos obtidos com a condição de parada 1.

Com relação à heurística MET, mesmo com todo o tempo de otimização utilizado, os tempos totais obtidos por ela foram ainda melhores que o da estratégia gulosa, gerada pela mesma heurística. Se o tempo máximo permitido (20 minutos) fosse menor, certamente os tempos totais também seriam menores. Podemos afirmar isso comparando os custos iniciais da heurística MET com os custos finais obtidos pelas outras heurísticas.

A Tabela 8 apresenta outros valores obtidos com as condições de parada 1 e 2.

Tabela 8 - Resultados comparativos

Heur.	Doc.	Custo Inicial (s)	Condição 1 (média)			Condição 2 (média)		
			Planos Avaliados	Custo Final (s)	Melhora	Planos Avaliados	Custo Final (s)	Melhora
gulosa	d33	4161,58	0	4161,58	0%	0	4161,58	0%
minmin	d33	16176,58	371074	**2535,41**	84%	1	4722,24	71%
maxmin	d33	8301,98	317721	2842,18	66%	9	3483,17	58%
MET	d33	**4161,58**	412560	2703,41	35%	412270	**2711,25**	35%
duplex	d33	8301,98	372574	2896,64	65%	56	3245,70	61%
gulosa	d65	8302,42	0	8302,42	0%	0	8302,42	0%
minmin	d65	20698,47	192016	6256,15	70%	22	8779,37	58%
maxmin	d65	15423,11	187499	**5188,41**	66%	12	6947,21	55%
MET	d65	**8302,42**	212196	5987,99	28%	212225	**5988,00**	28%
duplex	d65	15423,11	187778	5481,51	64%	73	6886,23	55%
gulosa	d129	17000,00	0	17000,00	0%	0	17000,00	0%
minmin	d129	32723,02	92369	11063,06	66%	30	15235,49	53%
maxmin	d129	28259,94	87669	10679,98	62%	23	13168,13	53%
MET	d129	**16953,60**	103437	12767,97	25%	103399	**12767,98**	25%
duplex	d129	28259,94	87535	**10568,68**	63%	30	13630,64	52%

Segundo a Tabela 8, podemos verificar que as soluções iniciais geradas pela heurística MET são menos custosas que as soluções iniciais geradas pelas outras heurísticas. Os valores são inclusive realmente próximos aos valores finais encontrados pelas outras heurísticas. Por isso a dificuldade em melhorá-la.

Segundo a mesma tabela, podemos perceber que é possível chegar a custos finais próximos aos custos obtidos com a condição de parada 1, analisando bem menos planos. A SLS-MC precisou analisar apenas um plano para reduzir à metade o custo da solução inicial gerada pela heurística min-min para o documento *d33*! Ou seja, através da heurística de aproximação utilizada na SLS-MC foi possível reduzir 50% dos custos

iniciais gerados pelas heurísticas de agendamento, analisando uma porção bem pequena do espaço de busca. Permitindo uma análise um pouco maior, como na condição 1, os planos podem ser melhorados ainda mais.

6.3.4 – Análise dos Trabalhos Relacionados à SLS-MC

As aplicações distribuídas como os *workflows* baseados em serviços Web e documentos AXML têm em comum o fato de atuarem em ambientes extremamente dinâmicos, regidos por um grande número de variáveis que determinam a configuração e o comportamento destas aplicações. Todo este dinamismo introduz várias dificuldades na escolha do plano de execução do *workflow* ou do plano de materialização do documento AXML. Este último acrescenta dificuldades ao contemplar grafos de dependências complexos, a possibilidade de delegação de tarefas em ambientes P2P e um conjunto específico de provedores para cada chamada de serviço Web.

A materialização de documentos AXML conforme abordada inicialmente, (ABITEBOUL, BONIFATI et al., 2003; MILO, ABITEBOUL et al., 2003a), garante a execução correta da materialização quanto à validade do tipo do documento resultante, porém faz a escolha do melhor sítio executor de forma gulosa, segundo uma análise de custos, no momento da materialização de cada serviço isolado. A materialização de documentos AXML não considera ainda a delegação de chamadas de serviços Web.

A geração de planos de materialização para documentos AXML pode ser comparada à otimização de consultas distribuídas envolvendo multi-junções, com a complexidade adicional do aninhamento de chamadas. Segundo (LANZELOTTE, VALDURIEZ et al., 1993), quando o número de relações é maior que 5 ou 6 e é necessário realizar multi-junções, a abordagem exaustiva se torna muito onerosa e os autores mostram experimentalmente a eficiência de estratégias heurísticas com uso de perturbação aleatória. Em (DOAN e HALEVY, 2002) é proposto um algoritmo para ordenação eficiente de planos de consultas sobre um esquema mediado. Para documentos AXML, ao contrário do que ocorre em (DOAN e HALEVY, 2002), não é possível aplicar uma função de custo monotônica, devido ao aninhamento das chamadas de serviço e à possibilidade de delegação das execuções.

No contexto do processamento de *workflows* em sistemas paralelos, existem vários algoritmos para o agendamento de tarefas de um grafo (ou seja, *list scheduling* e suas variações (BRAUN, SIEGEL et al., 2001; KWOK e AHMAD, 1999)). Tais algoritmos constituem a base das estratégias para distribuição de tarefas de *workflows*

em *Grids*. Contudo, essas estratégias são geralmente gulosas e consideram que as tarefas podem ser arbitrariamente atribuídas às máquinas, o que não ocorre com as chamadas de serviços de um documento AXML. Em (BLYTHE, JAIN et al., 1998), é proposto um método baseado em busca local para agendar a execução de *workflows* considerando todas as tarefas envolvidas. Esse método permite aceitar alternativas de desempenho inferior durante a geração do espaço de busca, visando escapar de mínimos locais. Já em (WU, SHU et al., 2001), o agendamento de *workflows* é baseado no algoritmo TASK, que explora a topologia dos nodos de um *workflow* para melhorar o desempenho da busca local. No TASK, o refinamento do plano é direcionado por mudanças seletivas que atingem principalmente nodos com alto custo. Essa idéia também é explorada no SLS-MC. Porém, tanto (WU, SHU et al., 2001) como (BLYTHE, JAIN et al., 1998) não consideram a delegação de partes do problema, além de pressuporem um *pool* de máquinas dedicadas para o agendamento de tarefas.

O otimizador XCraft (RUBERG e MATTOSO, 2005) visa a reduzir a complexidade total do problema e permitir uma melhor adaptação ao dinamismo dos ambientes P2P. Porém, uma dificuldade encontrada é a escolha da altura k dos sub-planos. É indesejável tanto a escolha de sub-planos muito pequenos quanto a escolha de sub-planos muito grandes. De qualquer maneira, em ambos os casos é possível que o XCraft gere grandes espaços de busca que podem levar a tempos de otimização inaceitáveis. A SLS-MC apresenta uma solução diretamente aplicável ao XCraft que minimiza o efeito do tamanho do espaço de busca ao aplicar a busca local estocástica e utilizar heurísticas pertinentes ao problema para avaliar estados vizinhos. Com a SLS-MC, é possível analisar seletivamente o espaço de buscas e evitar a busca exaustiva local, obtendo resultados melhores que os da busca exaustiva e gulosa conforme comprovam resultados apresentados. Com a SLS-MC foi possível obter custos finais até 50% mais baratos que os obtidos com a estratégia gulosa e tempos e otimização muito inferiores aos da solução exaustiva. O que pudemos perceber é que o resultado acumulado, isto é, o tempo total da SLS-MC, foi sempre menor que o da estratégia gulosa e sem comparação com o da exaustiva.

Para comprovar a não influência do tamanho do espaço de busca, os planos abstratos iniciais não foram quebrados em sub-planos menores e até mesmo os planos dos documentos com 129 chamadas foram analisados inteiramente. O que pudemos perceber, é que com um tempo pequeno de otimização, conseguimos atingir a "qualidade" almejada pelo usuário, analisando, na maioria das vezes, um número bem

pequeno de planos equivalentes. Ou ainda, com um mesmo tempo de otimização, conseguimos obter bons resultados para todos os documentos, independente de seu tamanho.

Assim, a SLS-MC contribui ao aplicar técnicas estocásticas a um problema atual que envolve a otimização de tarefas num grafo de dependências apresentando otimizações significativas ao estado da arte.

Capítulo 7 – Conclusão e Trabalhos Futuros

Este capítulo apresenta as conclusões obtidas com esta dissertação de mestrado. Ele faz um resumo do conteúdo descrito e apresenta as principais contribuições realizadas. Este capítulo apresenta ainda possíveis trabalhos futuros a serem desenvolvidos como evolução deste trabalho.

7.1 – Considerações Finais

Não existem mais dúvidas quanto às vantagens e benefícios obtidos pela utilização da linguagem XML e dos serviços Web na troca de informações e comunicação entre aplicações heterogêneas e distribuídas pela Web. A linguagem XML e os serviços Web constituem-se em poderosas ferramentas e logo foi possível perceber o grande potencial que poderia ser obtido ao utilizá-las em conjunto, como meios de integração de dados e de realização de tarefas seqüenciais.

Os documentos AXML surgiram como um resultado desta evolução. Eles correspondem a uma nova classe de documentos XML que possuem dados intencionais, representadas por chamadas a serviços Web. São documentos dinâmicos, utilizados para a integração de dados e serviços providos por vários sítios na Web. Podem ser vistos como *workflows* baseados em serviços Web, que aproveitam as potencialidades da linguagem XML e de serviços Web.

Um documento AXML pode ser composto por várias chamadas de serviço, cada qual com seus parâmetros de entrada. Através do aninhamento de chamadas de serviço e da definição de consultas XPath sobre o próprio documento é possível definir chamadas de serviços Web como parâmetros de entrada para outras chamadas de serviço Web. Estes parâmetros de entrada e ainda a existência de chamadas colaterais consistem nas dependências de execução de um documento AXML, e para que o conteúdo deste possa ser recuperado é preciso materializar (executar) todas as suas chamadas de serviço Web, respeitando-se as dependências existentes entre elas (*i.e.* as chamadas filho precisam ser executadas primeiro que as chamadas pai).

Os serviços Web demandados pelas chamadas de serviço podem ser executados por vários sítios na Web, bem como uma chamada pode ser delegada a outro sítio, para que este possa então materializá-la e retornar apenas os resultados relevantes. Estas características fazem com que um documento AXML possa ser materializado através de diversos planos equivalentes de materialização. O número de planos aumenta à medida que o número de chamadas, o número de sítios executores e delegados e o número de serviços Web aumenta. Trata-se de um problema NP - completo onde o uso de heurísticas torna-se obrigatório. Soluções utilizadas na distribuição de tarefas em sistemas paralelos e distribuídos e na otimização de consultas distribuídas, que são problemas parecidos ao da materialização de documentos AXML não se aplicam por não considerarem a delegação de chamadas e por normalmente englobarem estratégias exaustivas ou gulosas cujas soluções localmente ótimas, não garantem soluções globalmente ótimas. Diante da inexistência de soluções adequadas ao problema e inspirados por problemas de busca com espaços de busca muito grandes tratados através de algoritmos de busca local estocásticos como a Têmpera Simulada, por exemplo, nós propomos a SLS-MC. A SLS-MC trata-se de uma solução baseada em um algoritmo de busca local estocástico com múltiplas condições de parada para a geração eficiente de planos de materialização para documentos AXML.

As técnicas e algoritmos utilizados na SLS-MC foram escolhidas com o propósito de auxiliar o processo de geração eficiente de planos de materialização para documentos AXML de tal modo que fosse possível escolher um bom plano dentro de um tempo aceitável de otimização, levando-se em conta a volatilidade das redes P2P. A idéia principal era que o tempo de otimização pudesse ser controlado e pouco influenciado pelo tamanho do espaço de busca. Os algoritmos de busca local permitem que boas soluções sejam encontradas sem que todo o espaço de busca seja totalmente percorrido. Além disso, métodos com medidas estocásticas permitem uma busca mais eficiente já que evitam que a busca fique presa em um mínimo local ou platô e permitem a investigação de áreas distintas da topologia do espaço de busca.

As múltiplas condições de parada empregadas procuram evitar que seja desperdiçado muito tempo otimizando quando boas soluções, segundo critérios do próprio usuário, já tenham sido encontradas. Ao contrário do que acontece nas estratégias gulosas (que são extremamente rápidas, mas que acabam gerando planos ruins), pretendemos através da SLS-MC, encontrar soluções rapidamente, sem que esta "pressa" interfira de maneira significativa na qualidade dos planos encontrados.

Para avaliarmos adequadamente a SLS-MC através da realização de testes, nós desenvolvemos o SiMAX, um simulador para a plataforma Active XML que permite configurar facilmente sítios AXML e interligá-los por meio de redes P2P. No SiMAX também é possível configurar diferentes documentos AXML e serviços Web e distribuí-los pela topologia criada. Ainda é possível configurar o SiMAX com diferentes estratégias de busca, constituindo-se em um ambiente prático para a realização de testes e avaliação da SLS-MC.

Por meio dos testes realizados no SiMAX, a SLS-MC mostrou-se bastante eficiente, principalmente para documentos muito complexos. Para estes documentos, foi possível perceber uma diferença significativa em relação à solução gulosa. Na condição de parada 1, por exemplo, foi possível perceber uma melhora do tempo total de até 31% com a heurística duplex para o documento d129. Já para o documento d33, foi possível obter uma melhora de até 10% com a heurística min-min.

As condições de parada foram testadas separadamente, em conjunto com as heurísticas MET, min-min, max-min e duplex. Foi possível verificarmos através da comparação das condições de parada que as duas condições de parada possuem comportamentos complementares e convergem para resultados parecidos. Foi possível perceber ainda que a obtenção de bons planos de materialização não está atrelada ao tempo de otimização. Mesmo com baixos tempos de otimização é possível encontrar bons planos de materialização.

Por meio da combinação de diversas condições de parada, a SLS-MC pode parar assim que um bom plano for encontrado ou continuar até que o usuário ache que já percorreu suficientemente o espaço de busca.

Dentre as heurísticas analisadas, as que apresentaram um desempenho ligeiramente melhor foram a max-min e a MET. Esta última gerou soluções iniciais melhores, porém, difíceis de serem melhoradas. Para um mesmo tempo de otimização, a heurística MET acabou não gerando os melhores planos finais. Analisando, no entanto os tempos totais, verificamos que a heurística max-min apresentou uma ligeira vantagem, pois conseguiu bons tempos de otimização atrelados a bons tempos de execução. Os testes com as diferentes heurísticas evidenciaram que a SLS-MC, independentemente da qualidade da solução inicial, pode se recuperar e gerar bons planos através da análise seletiva do espaço de busca, sem precisar para isso de mais tempo de otimização. A SLS-MC conseguiu melhorar em até 70% a solução inicial gerada em alguns casos. Isso mostra que a SLS-MC é realmente melhor que qualquer

estratégia gulosa e que qualquer estratégia mais simples baseada em alguma dessas heurísticas. Por meio de testes comparativos com a estratégia exaustiva, ficou evidente também a superioridade da SLS-MC em relação a essa última. Com a condição de parada 1, em que o teto máximo dos 20 minutos foi atingido, ainda assim a melhora do tempo de otimização foi de mais de 1000% para todos os documentos!

Os testes com as duas condições de parada definidas na SLS-MC mostraram que é possível chegar a um acordo entre eficiência e qualidade. As duas, apesar de chegarem a resultados próximos, permitem comportamentos diferentes e se completam. Poderão juntas oferecer resultados mais satisfatórios aos usuários.

Finalmente, a SLS-MC representa uma solução adequada para o problema da geração eficiente de planos de materialização para documentos AXML, pois não se mostrou sensível ao tamanho do espaço de busca. A SLS-MC permite analisar seletivamente o espaço de busca, apresentando desta maneira bons tempos de otimização e ainda assim, bons planos de materialização.

7.2 – Contribuições desta Dissertação

a) Desenvolvimento de uma solução baseada em algoritmos de busca local estocástico para o problema da geração eficiente de documentos AXML

A principal contribuição desta dissertação de mestrado consiste na **SLS-MC**, uma solução que faz uso de algoritmos de **busca local estocásticos** associadas a múltiplas condições de parada para a geração eficiente de planos de materialização para documento AXML. A materialização de documentos AXML é considerada um problema NP - completo para o qual não existiam soluções prontas que atendessem a todos os requisitos apresentados nesta dissertação.

Através da SLS-MC conseguimos produzir planos de materialização de forma mais eficiente que as estratégias gulosa e exaustiva e vamos utilizá-la em conjunto com a estratégia iterativa e incremental do XCraft, acreditando ser possível obter resultados ainda melhores.

Embora aplicada no contexto dos documentos AXML, a SLS-MC pode ser aplicada de modo geral no contexto da otimização de *workflows* baseados em serviços Web, como os utilizados em malhas computacionais ou GRIDs. Em geral, tais aplicações também compreendem DAGs complexos que representam a execução de

serviços Web. No entanto, elas não consideram a delegação de chamadas e as funções de custo consideradas são diferentes. De qualquer maneira, a idéia da busca local estocástica, a geração de soluções iniciais baseadas em heurísticas de agendamento de tarefas e a análise dos nós que compõem o caminho crítico segundo uma heurística pertinente ao problema podem ser reaproveitadas e adaptadas.

b) Análise e adaptação de heurísticas de agendamento ao problema da materialização de documentos AXML

Com o objetivo de produzir soluções iniciais mais adequadas ao problema da materialização de documentos AXML, adaptamos o uso de quatro **heurísticas de agendamento** tradicionalmente utilizadas na alocação de tarefas em ambientes distribuídos (BRAUN, SIEGEL et al., 2001). Estas heurísticas são geralmente utilizadas para o agendamento de tarefas independentes e por isso houve a necessidade de adaptá-las ao contexto da materialização de documentos AXML, onde o aninhamento das chamadas de serviço impõe restrições e dependências de execução. As heurísticas foram utilizadas segundo a topologia dos planos, e somente tarefas independentes e concorrentes, cujos filhos já haviam sido analisados, podiam ser analisados pelas heurísticas. As heurísticas adaptadas para gerar as soluções iniciais na SLS-MC foram: **MET, Min-min, Max-min e Duplex**.

c) Introdução de múltiplas condições de parada

Ao contrário de muitas meta-heurísticas e métodos utilizados em problemas de busca, que geralmente são executados por um número fixo de iterações ou até que um mínimo local seja atingido, a SLS-MC procurou introduzir o conceito de **múltiplas condições de parada** com o objetivo de permitir ao usuário definir seus anseios de várias maneiras, através de um **ganho de custo e de tempo**. Por meio destas duas condições de parada o usuário pode estabelecer o "acordo" entre qualidade e eficiência que ele almeja sem a necessidade de aguardar por um tempo fixo (e muitas vezes longo), quando na verdade uma solução suficientemente boa (segundo seus critérios) já foi encontrada.

Como medida de segurança, já que estas condições de parada podem representar ainda um tempo bastante significativo de otimização, foram definidos tetos máximos,

ou seja, condições de parada de segurança que impedem que a busca prossiga indefinidamente. As condições de segurança definidas foram: **número máximo de reinícios**, que permitem identificar que dificilmente será possível encontrar uma solução melhor a já encontrada, **tempo máximo** de otimização e **número máximo de planos** a serem investigados. Quando qualquer uma das condições de parada enunciadas é atingida, a busca é interrompida e o melhor plano obtido até o momento é o escolhido.

d) Desenvolvimento de um simulador para a realização de testes com a SLS-MC

A característica distribuída e heterogênea das redes P2P aliadas à dificuldade de acesso aos sítios que compõem a rede dificultam a execução de testes e análise da solução proposta. Visando transpor estes problemas desenvolvemos um simulador batizado de **SiMAX** que permite avaliar não só a SLS-MC no contexto da materialização de documentos AXML, mas também comparar seus resultados com os resultados de outras estratégias de otimização também abordadas pelo XCraft.

O SiMAX foi desenvolvido sobre a plataforma Active XML, cujo componente otimizador XCraft sofreu modificações que permitiram agregar a estratégia SLS-MS e suas heurísticas de agendamento na plataforma Active XML.

Por meio de propriedades definidas pelo usuário, o SiMAX permite analisar diferentes estratégias de otimização, segundo diferentes configurações de redes P2P, serviços Web e documentos AXML. O perfil de otimização do SiMAX é que determina a estratégia de otimização que será utilizada (gulosa, exaustiva ou SLS-MC) e suas configurações segundo as quais os testes são executados. O SiMAX produz ainda relatórios que apresentam os diversos resultados obtidos com os testes realizados.

7.3 – Trabalhos Futuros

Apesar de a SLS-MC ter apresentado bons resultados e se mostrado bastante eficiente no processo de materialização de documentos AXML, acreditamos que possível melhorá-la ainda mais. Uma perspectiva para evoluir a proposta do SLS-MC consiste na especificação de uma versão distribuída da estratégia, onde o espaço de busca é quebrado e distribuído entre sítios para que a otimização, e não só a execução, possa ser delegada e, portanto otimizada. Para tal, a estratégia SLS-MC deveria ser

implementada por cada sítio da rede e realizar a otimização segundo os critérios configurados pelo usuário. Ao término da otimização, os sítios enviam os melhores planos para o sítio mestre que dispara a materialização e consolida os resultados.

Para a realização de testes com a evolução da SLS-MC, seria preciso também evoluir também o SiMAX, de tal modo que os sítios configurados possam receber partes do espaço de busca e executar a busca concorrentemente. Para resultados mais coerentes, o SiMAX deverá ser distribuído, ou seja, executado em máquinas distintas, para que os tempos de otimização não sofram degradações por executarem em uma única máquina.

Seria interessante também realizar novos testes que averiguem melhores valores de probabilidade de aceitação para planos ruins. Nesta dissertação, assumimos 50% de aceitação, já que nosso principal objetivo era verificar a eficiência da SLS-MC. No entanto, com o valor de aceitação melhor ajustado, acreditamos ser possível melhorar ainda mais os resultados obtidos.

Referências Bibliográficas

AALST, W. M. P. V. D., 2003, "Don't Go With the Flow: Web Services Composition Standards Exposed", *IEEE Intelligent Systems*, v. 18, n. 1, pp. 72-76

AALST, W. M. P. V. D., HOFSTEDE, A. H. M. K. B., BARROS, A. P., 2003, "Workflow Patterns", *Distributed and Parallel Databases*, v. 14, n. 1, pp. 5-51

ABITEBOUL, S., BENJELLOUN.O., MANOLESCU, I., et al, 2002, "Active XML: Peer-to-Peer Data and Web Services Integration". In: *Very Large Databases*, pp. 1087-1090

ABITEBOUL, S., BENJELLOUN.O., MILO, T., 2004, "Positive Active XML". In: *ACM PODS*, pp. 35-45

ABITEBOUL, S., BONIFATI, A., COBENA, G., et al, 2003, "Dynamic XML Documents with Distribution and Replication". In: *ACM SIGMOD*, pp. 527-538

ABITEBOUL, S., MANOLESCU, I., MILO, T., et al, 2004, "Active XML: A Data-Centric Perspective on Web Services". In: *Web Dynamics*, pp. 275-300

APACHE JELLY, 2006, "Apache Jelly: Executable XML".http://jakarta.apache.org/commons/jelly

ARKIN, A., INTALIO, 2002, "Business Process Modeling Language (BPML)"

ARKIN, A., ASKARY, S., FORDIN, S., et al, 2002, "Web Service Choreography Interface (WSCI) 1.0". In: http://www.w3.org/TR/wsci/.

AZEVEDO, V., 2004, *WebTransact-EM: Um Modelo Para a Execução Dinâmica de Serviços Web Semanticamente Equivalentes.*, Tese de Mestrado, Universidade Federal do Rio de Janeiro

BEA, IBM, MICROSOFT, et al, 2003, *BPEL4WS. Business Process Execution Language for Web Services Version 1.1*.http://www-106.ibm.com/develpernetworks/webservices/library/ws-bpel

BELKHALE, K. P.,BANERJEE, P., 1991, "A Scheduling Algorithm for Parallelizable Dependent Tasks". In: *International Parallel Processing Symposium*, pp. 500-506

BLUM, C.,ROLI, A., 2003, "Metaheuristics in Combinatorial Optimization: Overview and Conceptual Comparison", *ACM Computing Surveys*, v. 35, n. 3, pp. 268-308

BLYTHE, J., JAIN, S., DEELMAN, E., et al, 2005, "Task Scheduling Strategies for Workflow-based Applications in Grids". In: *International Symposium on Cluster Computing and the Grid*, pp. 759-767

BLYTHE, J., JAIN, S., DEELMAN, E., et al, 1998, "Task Scheduling Strategies for Workflow-based Applications in Grids"

BOX, D., EHNEBUSKE, D., KAKIVAYA, G., et al, 2000, "Simple Object Access Protocol (SOAP) 1.1". In: http://www.w3.org/TR/2000/NOTE-SOAP-20000508/.

BRAUN, T. D., SIEGEL, H. J., BECK, N., et al, 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, pp. 810-837

BRAY, T., PAOLI, J., SPERBERG-MCQUEEN, M., et al, 2004, "Extensible Markup Language (XML) 1.0 (Third Edition)". In: http://www.w3.org/TR/REC-xml/.

BUSINESS PROCESS PROJECT TEAM, 2001, "ebXML Business Process Specification Schema Version 1.01".

CASANOVA, H., LEGRAND, A., ZAGORODNOV, D., et al, 2000, "Heuristics for Scheduling Parameter Sweep Applications in Grid Environments". In: *Heterogeneous Computing Workshop*, pp. 349-363

CHRISTENSEN, E., CURBERA, F., MEREDITH, G., et al, 2001, "Web Services Description Language (WSDL) 1.1". In: http://www.w3.org/TR/wsdl.html.

COLORNI, A., DORIGO, M., MANIEZZO, V., et al, 2006, "Heuristics From Nature for Hard Combinatorial Optimization Problems", *International Transactions in Operational Research*, v. 3, n. 1, pp. 1-21

CORMEN, T. H., LEISERSON, C. E., RIVEST, R. L., 2001, *Algorithms*. 2 ed, The MIT Press

DOAN, A.,HALEVY, A. Y., 2002, "Efficiently Ordering Query Plans for Data Integration". In: *International Conference on Data Engineering*, pp. 393-402

EXTENSIBLE MARKUP LANGUAGE (XML), 2006. In: http://www.w3.org/xml.

FREUDER, E. C., NAREYEK, A., FOURER, R., et al, 2005, "Constraints and AI Planning", *IEEE Intelligent Systems*, v. 20, n. 2, pp. 62-72

GLOVER, F., 1986, "Future Paths for Integer Programming and Links to Artificial Intelligence", *Computers & Operations Research*, v. 13, n. 5, pp. 533-549

GRID COMPUTING, 2006. In: http://www.grid.org.

HERTZ, A., TAILLARD, E., WERRA, D., 2006, "A Tutorial on Tabu Search". In: http://www.cs.colostate.edu/~whitley/CS640/hertz92tutorial.pdf.

HORST, R., HOANG, T., TUY, H., 1996, *Global Optimization: Deterministic Approaches*. Springer

JAKARTA COMMONS DIGESTER, 2006, "Commons Digestger". In: http://jakarta.apache.org/commons/digester/.

KIRKPATRICK, S., GELATT, C. D., VECCHI, M. P., 1983, "Optimization by Simulated Annealing", *Science*, v. 220, n. 4598, pp. 671-680

KWOK, Y.-F.,AHMAD, I., 1999, "Static Scheduling Algorithms for Allocating Directed Task Graphs to Multiprocessors", *ACM Computing Surveys*, v. 31, n. 4, pp. 406-471

LANZELOTTE, R. S. G., VALDURIEZ, P., ZAIT, M., 1993, "On the Effectiveness of Optimization Search Strategies for Parallel Execution Spaces". In: *Very Large Data Bases*, pp. 493-504

LEYMANN, F., 2001, "Web Services Flow Language Version 1.0". In: http://www-306.ibm.com/software/solutions/webservices/pdf/WSFL.pdf.

MEYER, L. A. V. C., 2006, *Estratégias para o Escalonamento Dinâmico de Workflows em Grid*, Tese de Doutorado, Universidade Federal do Rio de Janeiro

MILO, T., ABITEBOUL, S., AMANN, B., et al, 2003a, "Exanging Intensional XML Data"

MILO, T., ABITEBOUL, S., AMANN, B., et al, 2003b, "Exchanging Intensional XML Data". In: *ACM SIGMOD Conference*, pp. 289-300

MILOJICIC, D. S., KALOGERAKI, V., LUKOSE, R., et al, 2002, *Peer-to-Peer Computing*. HP Laboratories

NG, W. S., OOI, B. C., TAN, K. Z. A., 2003, "PeerDB: A P2P-based System for Distributed Data Sharing". In: *International Conference on Data Engineering*, pp. 633-644

OLIVER SINNEN,LEONEL SOUSA, 2001, "Scheduling Task Graphs on Arbitrary Processor Architectures Considering Contention". In: *High-Performance Computing and Networking (Europe)*, pp. 373-382

PEREIRA, D., RUBERG, G., MATTOSO, M., 2006, "Geração Eficiente de Planos de Materialização para Documentos AXML". In: *XXI Simpósio Brasileiro de Banco de Dados*, pp. 236-250

PEREIRA, D., RUBERG, G., MATTOSO, M., 2005, "Geração Eficiente de Planos de Materialização para Documentos AXML". In: *Workshop de Teses e Dissertações em Banco de Dados - SBBD*

RESENDE, M. G. C.,RIBEIRO, C. C., 2001, *Greedy Randomized Adaptive Search Procedures*. AT&T Labs

RUBERG, G., MATTOSO, M., 2005, "XCraft, A Dynamic Optimizer for the Materialization of Active XML Documents"

RUBERG, N., RUBERG, G., MANOLESCU, I., 2004, "Towards Cost-based Optimization for Data-intensive Web Service Computations". In: *XIX Simpósio Brasileiro de Banco de Dados*, pp. 283-297

RUSSEL, S.,NORVIG, P., 2003, *Inteligência Artificial*. 2 ed. São Paulo, Campus

SAKELLARIOU, R.,ZHAO, H., 2004, "A Hybrid Heuristic for DAG Scheduling on Heterogeneous Systems". In: *International Parallel and Distributed Processing Symposium*, Manchester

SCHOPF, J. M.,BERMAN, F., 1999, "Stochastic Scheduling", *SC*, pp. 48-

SCRIPTOR, G. S., 2006, *Metaheuristics and Combinatorial Optimization Problems*, Tese de Mestrado, University of Buffalo

STUTZLE, T., 1998a, *Local Search Algorithms for Combinatorial Problems - analysis, algorithms,and new applications.*

STUTZLE, T., 1998b, *Local Search Algorithms for Combinatorial Problems - Analysis, Improvements, and New Applications.*, Tese de Doutorado, Universit¨at Darmstadt

TATARINOV, I., IVES, Z. G., MADHAVAN, J., et al, 2003, "The Piazza Peer Data Management Project.", *ACM SIGMOD Record*, v. 32, n. 2, pp. 47-52

TAYLOR, I., SHIELDS, M., WANG, I., et al, 2003, "Triana Applications within Grid Computing and Peer to Peer Environments", *Journal of Grid Computing*, v. 1, pp. 199-217

THATTE, S., 2001, "XLANG: Web Services for Business Process Design". In: http://www.gotdotnet.com/team/xml_wsspecs/xlang-c/default.htm.

THE BRANCH AND BOUND APPROACH, 2006. In: http://www.csulb.edu/~obenli/Research/IE-encyc/bb.html.

TOWARDS OPEN GRID SERVICES ARQUITECTURE, 2006. In: www.globus.org/ogsa.

UDDI.ORG, 2006, "UDDI Tecnical White Paper".Disponível em: http://www.uddi.org/pubs/Iru_UDDI_Technical_White_Paper.PDF

VALDURIEZ, P.,ÖZSU, M. T., 2001, *Princípios de Sistemas de Banco de Dados Distribuídos*. 2 ed. Rio de Janeiro, Campus

VARGAS, P. K., SANTOS.L.A.S., SCHULZE, B., 2005, "Hierarchical Submission in Grid Environment". In: *International Workshop on Middleware for Grid Computing*

VIRDELL, M., 2003, "Business Process and Workflows in the Web".http://www-128.ibm.com/developerworks/webservices/library/ws-work.html

WEB SERVICES ACTIVITY, 2006. In: http://www.w3.org/WS.

WFMC, 1999, "Workflow Management Coalition - Terminology & Glossary. Specification WFMC-TC-1011". In: http://www.wfmc.org/standards/docs/TC-1011_term_glossary_v3.pdf.

WIECZOREK, M., PRODAN, R., FAHRINGER, T., 2005, "Scheduling of scientific workflows in the ASKALON grid environment", *ACM SIGMOD Record*, v. 34, n. 3, pp. 56-62

WU, M., SHU, W., GU, J., 2001, "Efficient Local Search for DAG Scheduling", *IEEE Transactions on Parallel and Distributed Systems*, v. 12, n. 6, pp. 617-627

XML, 2006. In: http://www.w3.org/XML/.

YU, J.,BUYYA, R., 2005, "A Taxonomy of Workflow Management Systems for Grid Computing", *Journal of Grid Computing*, v. 3, n. 3-4, pp. 171-200