



GERÊNCIA DE PROVENIÊNCIA EM WORKFLOWS CIENTÍFICOS
PARALELOS E DISTRIBUÍDOS

Luiz Manoel Rocha Gadelha Júnior

Tese de Doutorado apresentada ao Programa de Pós-graduação em Engenharia de Sistemas e Computação, COPPE, da Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Doutor em Engenharia de Sistemas e Computação.

Orientador: Marta Lima de Queirós Mattoso

Rio de Janeiro
Agosto de 2012

GERÊNCIA DE PROVENIÊNCIA EM WORKFLOWS CIENTÍFICOS
PARALELOS E DISTRIBUÍDOS

Luiz Manoel Rocha Gadelha Júnior

TESE SUBMETIDA AO CORPO DOCENTE DO INSTITUTO ALBERTO LUIZ
COIMBRA DE PÓS-GRADUAÇÃO E PESQUISA DE ENGENHARIA (COPPE)
DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS
REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE DOUTOR
EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

Examinada por:

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

Prof. Alexandre de Assis Bento Lima, D.Sc.

Prof^a. Ana Tereza Ribeiro de Vasconcelos, D.Sc.

Prof. Fábio André Machado Porto, D.Sc.

Prof. Leonardo Gresta Paulino Murta, D.Sc.

Prof. Nelson Francisco Favilla Ebecken, D.Sc.

RIO DE JANEIRO, RJ – BRASIL
AGOSTO DE 2012

Gadelha Júnior, Luiz Manoel Rocha

Gerência de Proveniência em Workflows Científicos Paralelos e Distribuídos/Luiz Manoel Rocha Gadelha Júnior. – Rio de Janeiro: UFRJ/COPPE, 2012.

XI, 104 p.: il.; 29,7cm.

Orientador: Marta Lima de Queirós Mattoso

Tese (doutorado) – UFRJ/COPPE/Programa de Engenharia de Sistemas e Computação, 2012.

Referências Bibliográficas: p. 43 – 62.

1. Proveniência. 2. Workflows Científicos. 3. Computação Paralela e Distribuída. 4. Segurança da Informação. I. Mattoso, Marta Lima de Queirós. II. Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia de Sistemas e Computação. III. Título.

*À minha mulher, Carolina, e ao
meu filho, Eduardo. Aos meus
pais, Maria Juvenha e Luiz
Manoel, e às minhas irmãs,
Melissa e Bruna.*

Agradecimentos

À minha mulher, Carolina, e ao meu filho, Eduardo, pelo amor, apoio e paciência, fundamentais para a realização deste trabalho.

Aos meus pais, Maria Juvenha e Luiz Manoel, pela educação, lições de vida e apoio.

Às minhas irmãs, Melissa e Bruna, pelo companheirismo.

Aos meus sogros, Denise e Cláudio, pelo apoio em diversas ocasiões.

À Prof^a. Marta Mattoso, pela excelente orientação, dedicação e incentivo durante este trabalho.

Ao Prof. Ian Foster e seu grupo, por me receberem durante o estágio sanduíche e pela colaboração realizada.

Ao Michael Wilde pelo tempo, dedicação, diversas sugestões e por ter me apresentado a usuários do Swift, fundamentais na parte experimental deste trabalho.

Ao Prof. Alexandre Assis, à Prof^a. Ana Tereza Vasconcelos, ao Prof. Fábio Porto, ao Prof. Leonardo Murta e ao Prof. Nelson Ebecken por aceitarem integrar a banca de examinadores.

Ao Programa de Engenharia de Sistemas e Computação pelo excelente ambiente acadêmico.

Ao LNCC, pela liberação para realização deste trabalho.

À CAPES, pelo financiamento do meu estágio sanduíche na Universidade de Chicago.

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

GERÊNCIA DE PROVENIÊNCIA EM WORKFLOWS CIENTÍFICOS PARALELOS E DISTRIBUÍDOS

Luiz Manoel Rocha Gadelha Júnior

Agosto/2012

Orientador: Marta Lima de Queirós Mattoso

Programa: Engenharia de Sistemas e Computação

A pesquisa científica é cada vez mais apoiada por experimentos computacionais, que são frequentemente dados por um grande número de tarefas computacionais (MTC, *many-task computing*) que são especificadas e automatizadas como workflows científicos. Nesta tese, abordamos o problema do gerência de informações de proveniência sobre computações que envolvam muitas tarefas (MTC), implementadas através de workflows científicos, e que sejam realizadas em ambientes paralelos e distribuídos de alto desempenho. Neste contexto de larga escala, as questões envolvidas no gerenciamento de proveniência tornam-se mais complexas pois os mecanismos utilizados na coleta e armazenamento dessas informações podem ter um impacto que prejudique a escalabilidade do experimento computacional. Por outro lado, o nível de detalhe das informações capturadas deve ser suficiente para permitir a análise do experimento de maneira satisfatória. Além de permitir a análise do processo de derivação de dados de um experimento, a proveniência pode apoiar a depuração e otimização de estratégias de execução e de manuseio de dados de workflows, se as respectivas informações relativas a consumo de recursos computacionais forem adicionadas. Desenvolvemos o MTCProv, um arcabouço para consultas de proveniência que captura os detalhes do tempo de execução em sistemas paralelos e distribuídos, além da proveniência prospectiva e retrospectiva padrões. Uma interface de consulta esconde as complexidades da consulta relacional, mas permite associar proveniência com detalhes do paralelismo e acesso aos dados do domínio da aplicação. Avaliamos o MTCProv utilizando uma aplicação de modelagem de proteínas e descrevemos como os padrões de consulta identificados neste trabalho são expressos de forma simples. Resultados de desempenho mostram escalabilidade com excelente relação custo-benefício ao coletar a proveniência.

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

PROVENANCE MANAGEMENT IN PARALLEL AND DISTRIBUTED SCIENTIFIC WORKFLOWS

Luiz Manoel Rocha Gadelha Júnior

August/2012

Advisor: Marta Lima de Queirós Mattoso

Department: Systems Engineering and Computer Science

Scientific research is increasingly assisted by computer-based experiments. Such experiments are often composed of a vast number of computational tasks that are specified and automated as scientific workflows. This large scale is also characteristic of the data that flows within such “many-task” computations (MTC). Provenance information can record the behavior of such computational experiments via the lineage of process and data artifacts. In this thesis, we address the problem of managing provenance information of MTC, implemented as scientific workflows, and carried out in high performance parallel and distributed environments. In this large scale context, the issues involved in managing provenance become more complex since the mechanisms used to collect and store this information may have a detrimental impact on the scalability of the computational experiment. However, the level of detail of the captured information should be enough to enable adequate experiment analysis. In addition to allowing the analysis of the data derivation process of an experiment, provenance can support debugging and optimization of execution strategies and workflow data management, if information on computational resource consumption is added. We developed MTCProv, a provenance query framework that captures runtime details in parallel and distributed systems, and standard prospective and retrospective provenance. A query interface hides the complexities of relational querying, but allows for associating provenance with details of parallelism and access to application domain data. We evaluate MTCProv using a protein modeling application and describe how the query patterns of identified in this work are expressed in a simple way. Performance results show scalability with excellent cost-benefit while collecting provenance.

Sumário

Lista de Figuras	x
Lista de Tabelas	xi
1 Introdução	1
1.1 Proveniência de Workflows Paralelos e Distribuídos	4
1.2 Definição do Problema	6
1.3 Organização da Tese	8
2 Conceitos Preliminares	10
2.1 Experimentos Científicos Computacionais	10
2.2 Workflows Científicos	11
2.3 Proveniência	16
2.4 Swift	17
2.5 Segurança da Informação	22
3 Trabalhos Relacionados	25
3.1 Modelos de Proveniência	25
3.2 Análise de Tempo de Execução	26
3.3 Consultas	27
3.4 Segurança	28
4 Gerência de Proveniência em Workflows Científicos Paralelos e Dis-	
tribuídos	32
4.1 Modelagem de Proveniência	32
4.2 Gerência Escalável de Proveniência em Workflows Científicos Parale-	
los e Distribuídos	33
4.3 Aspectos de Segurança de Proveniência	36
5 Discussão e Conclusões	39
5.1 Perspectivas Futuras	41
Referências Bibliográficas	43

A	Provenance Management in Many-Task Computing	63
A.1	Data Model	63
A.2	PC3: Implementation and Queries	67
A.3	PC3: Evaluation	74
B	MTCProv: A Practical Provenance Query Framework for Many-Task Scientific Computing	77
B.1	Provenance Query Patterns	77
B.2	An Extended Provenance Model for Many-Task Scientific Computations	81
B.3	Design and Implementation of MTCProv	86
B.3.1	Provenance Gathering and Storage	86
B.3.2	Query Interface	89
B.4	Case Study: Protein structure prediction workflow	92
C	Security Aspects of Provenance Systems for Distributed Scientific Workflows	96
C.1	Security Requirements for Provenance Systems	96
C.2	Kairos: Protecting Provenance Authorship and Temporal Information	97
C.2.1	Kairos: Secure Provenance Architecture	98
C.2.2	Kairos Overview	98
C.2.3	Kairos Implementation Issues	102
C.2.4	Demonstration	103

Lista de Figuras

1.1	Atividades do DES a partir de um mosaico de imagens do céu [1].	2
2.1	Ciclo de vida de um experimento científico computacional [2].	10
2.2	Composição gráfica de workflow: Vistrails [3]. Fonte: http://www.vistrails.org	13
2.3	Padrão de controle de fluxo: sequência.	14
2.4	Padrão de controle de fluxo: bifurcação paralela.	14
2.5	Padrão de controle de fluxo: sincronização.	15
2.6	Padrão de controle de fluxo: laço.	15
2.7	Arestas do OPM.	17
2.8	Descrições distintas de uma mesma execução. Fonte: Moreau et al. [4]	18
2.9	Protocolo de datação criptográfica TSP.	24
4.1	Implementação inicial do workflow de paralelização do BLAST.	34
4.2	Refinamento do workflow de paralelização do BLAST.	36
A.1	Provenance database schema.	65
A.2	Provenance relationships of a <code>sortProg</code> execution.	66
A.3	LoadWorkflow. Source: PC3 web site [5].	69
B.1	Protein structure prediction workflow.	80
B.2	UML diagram of Swift's provenance database.	84
B.3	Examples of annotations enabled by MTCProv.	85
B.4	MTCProv components and flow	87
B.5	Impact of gathering provenance.	88
B.6	Higher-level schema that summarizes the underlying provenance da- tabase schema.	90
B.7	Plots of query outputs.	94
C.1	Kairos: interaction between various PKI components.	99
C.2	Kairos: procedure overview.	100

Lista de Tabelas

3.1	Comparação de abordagens para segurança de proveniência.	30
A.1	Database relation <code>processes</code>	64
A.2	Database relation <code>dataset_usage</code>	64
A.3	Equivalence between tuples in the <code>dataset_usage</code> table and OPM. . .	67
A.4	LoadWorkflow Activities - Pre-Load Section	70
A.5	LoadWorkflow Activities - Load Section	71
A.6	LoadWorkflow Activities - Post-Load Section	71
B.1	Provenance query patterns found in the Provenance Challenge series.	79
C.1	Applicable security standards.	102

Capítulo 1

Introdução

A pesquisa científica e o desenvolvimento tecnológico têm se apoiado em métodos numéricos e simulações executadas em computadores desde a criação dos mesmos. O surgimento de computadores paralelos, redes de alta velocidade e da computação distribuída aumentou a capacidade computacional disponível consideravelmente, permitindo que problemas cada vez mais intensivos computacionalmente fossem tratados. Mais recentemente, a proliferação de sensores científicos, como telescópios, aceleradores de partículas, e sequenciadores de genoma causou um aumento exponencial do volume de dados manipulados por computações científicas. Exemplos que evidenciam este aumento incluem:

- O NCBI (*National Center for Biotechnology Innovation*) mantém a base de dados GenBank, de sequências de nucleotídeos, que tem dobrado de tamanho a cada 18 meses. Em Abril de 2012, o GenBank [6] possuía aproximadamente 140 bilhões de bases. Embora o volume de dados não seja relativamente grande, cerca de 1 Terabyte, o processamento destes é computacionalmente intensivo, envolvendo algoritmos como alinhamento de sequências, construção de árvores evolutivas e determinação de estrutura tridimensional de proteínas [7].
- O LHC (*Large Hadron Collider*) é um acelerador de partículas em formato de anel com 27km de circunferência localizado entre a Suíça e a França. O objetivo do experimento é detectar partículas subatômicas, como o bóson de Higgs, cuja existência, ainda não comprovada, é prevista nos modelos teóricos da Física. O principal detector montado no LHC, chamado ATLAS, produz cerca de 15PB de dados brutos por ano, que são processados por uma infraestrutura de computação distribuída, composta por centros de computação localizados em cerca de 30 países, em busca dos padrões, previstos na teoria, que comprovariam a detecção das partículas. Em 4 de julho de 2012 foi anunciada [8] a detecção de uma partícula consistente com o modelo teórico do bóson de Higgs.

- O DES (*Dark Energy Survey*), cujas atividades são ilustradas na figura 1.1, é um projeto para rastrear a aceleração da expansão do universo, que seria explicada pela atuação da *energia escura*. Uma câmera de 570 megapixels montada em um telescópio no Chile captura 400 imagens, totalizando 6,6 Terabytes, por noite. Uma infraestrutura de computação distribuída [1] foi montada para o processamento dessas imagens, que necessitarão de 10 milhões de horas de processamento e 4 Petabytes de armazenamento ao longo do projeto.

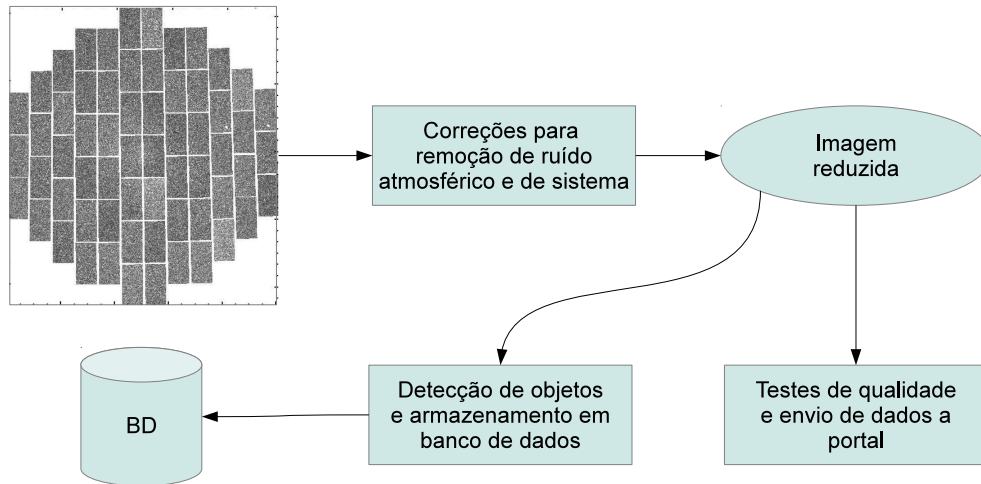


Figura 1.1: Atividades do DES a partir de um mosaico de imagens do céu [1].

- O SKA (*Square Kilometre Array*) [9, 10], cuja construção será iniciada em 2017 e concluída em 2024, será composto por 3.000 rádio-telescópios de 15m de diâmetro. O projeto reunirá cerca de 70 instituições de 20 países. Estima-se que o volume de dados brutos gerado diariamente será de 1 Exabyte, cerca de duas vezes o volume de dados que trafega pela Internet por dia atualmente. O dados gerados após o processamento dos dados brutos necessitarão de 300 Petabytes a 1,5 Exabyte de armazenamento permanente por ano.

Tais projetos requerem infraestruturas adequadas para *computação intensiva em dados*, que pode ser definida como qualquer tarefa computacional na qual a disponibilidade de dados seja o fator limitante na geração de soluções a uma taxa aceitável [11]. Szalay et al. [12] propõem o uso de recursos computacionais nos quais a relação entre a capacidade de processamento e a vazão de dados seja mais balanceada que a de supercomputadores tradicionais. Neste contexto, é comum que uma única instituição não atenda, isoladamente, aos requisitos destes experimentos científicos computacionais. Requer-se então o uso de infraestruturas de computação distribuída,

como as *grades computacionais* [13], que permitem que diversas instituições agreguem seus recursos computacionais, ou a *computação na nuvem* [14–16], que utiliza técnicas de virtualização para o provisionamento, de forma rápida, dinâmica e escalável, de recursos computacionais, plataformas de desenvolvimento ou aplicações.

Uma questão importante em experimentos computacionais em larga escala é a gerência de dados [17–20]. Como observado por Deelman e Chervenak [19], cientistas podem ter acesso a recursos computacionais de alto desempenho. No entanto, a transferência para esses recursos dos dados requeridos por suas computações é propensa a erros e pode ser dispendiosa em termos de tempo. Normalmente esses dados são mantidos em sistemas de armazenamento distantes dos recursos computacionais, que geralmente têm capacidade limitada de armazenamento, e têm que ser transferidos antes da execução. Chervenak et al. [17] apresentaram uma arquitetura para gerência de dados em grade e descreveram uma série de serviços oferecidos por esses sistemas, entre os quais, um serviço de armazenamento e acesso aos dados que permita a agregação de espaço de armazenamento distribuído e heterogêneo, a gerência de réplicas de arquivos e o acesso a arquivos de forma transparente para os usuários. Exemplos de sistemas que fornecem estes serviços incluem o SRB [21] e o iRods [22], sistemas de arquivos distribuídos, que permitem que usuários presentes em diferentes locais visualizem a mesma estrutura de diretórios e arquivos. O GridFTP [23] e RLS [24] são componentes do Globus Toolkit [25], um conjunto de ferramentas para implementação de grades computacionais, e permitem a transferência de grandes massas de dados entre recursos computacionais e a gerência de réplicas de conjuntos de dados. Mais recentemente, o Globus Online [26] foi desenvolvido como um serviço web para transferências de dados entre sítios de computação distribuídos.

Diversos autores [11, 27–31] propõem que a análise dos dados destes experimentos computacionais pode ser considerada um quarto paradigma científico, em adição aos tradicionais paradigmas experimental, analítico e de simulações computacionais. Gray [28] observa que não existem ferramentas que permitam apoiar, de forma integrada, todas as etapas envolvidas nestes experimentos, como a coleta de dados brutos nos sensores, a curadoria dos mesmos, a execução de análises, a estruturação de dados para consultas, a visualização, e a publicação científica em formato que permita que leitores e revisores acessem os dados utilizados e reproduzam os experimentos computacionais de forma automatizada. O termo *e-Ciência* [32, 33] é frequentemente utilizado para denotar atividades deste tipo.

A gerência manual de experimentos científicos computacionais em larga escala é trabalhosa e sujeita a erros pois diversos aspectos tem que ser levados em consideração, por exemplo:

- as atividades podem ter dependências de dados entre si, de modo que algumas atividades só podem iniciar a sua execução quando seus dados de entrada,

produzidos por outras atividades que as precedem, estiverem disponíveis;

- eventualmente o fluxo de execução do experimento pode sofrer uma bifurcação para a execução de diversas atividades independentes entre si, tornando interessante a execução paralela das mesmas por questões de eficiência;
- o início da execução de uma atividade pode depender do fim da execução de diversas atividades disparadas após uma bifurcação, requerendo uma sincronização da execução do experimento, onde é necessário garantir que todas as atividades geradas pela bifurcação de fato terminaram a sua execução;
- caso o experimento utilize diversos recursos computacionais remotos, é necessário gerenciar a transferência de dados e monitorar a execução de atividades remotas.

Como observado por Davidson e Freire [34], esses aspectos são difíceis de se controlar quando são utilizadas linguagens de *scripting* genéricas como Perl ou Python, populares em áreas como a bioinformática [7]. A automatização destes experimentos pode ser apoiada por sistemas de gerência de workflows científicos (SGWCs) [35–39], que permitem projetar, executar e analisar experimentos dados pela composição de diversas aplicações científicas, possivelmente distintas, que são encadeadas através da produção e consumo de dados. De acordo com Deelman et al. [38], a utilização de SGWCs permite que os cientistas se concentrem no experimento científico ao invés de dedicarem uma parcela do seu tempo à gerência da computação. Sistemas de gerência de workflows [40] são utilizados também em ambientes corporativos. Barga e Gannon [41] observam que em ambientes corporativos é enfatizada a orquestração de processos de negócio, que frequentemente não envolvem computação, enquanto no contexto científico a ênfase é dada à orquestração de aplicações científicas.

1.1 Proveniência de Workflows Paralelos e Distribuídos

A análise de experimentos computacionais também se beneficia de automatização, devido à grande quantidade de informação que geralmente é produzida nos mesmos. Informações de *proveniência* [34, 42–51], que reúnem detalhes sobre a concepção e a execução desses experimentos, descrevendo os processos e dados envolvidos na geração dos resultados dos mesmos, podem ser utilizadas para facilitar esta tarefa. Elas permitem a descrição precisa de como o experimento foi projetado, chamada de proveniência *prospectiva*, e do que ocorreu durante sua execução, denominada proveniência *retrospectiva*. Algumas aplicações da proveniência incluem a reprodução de um experimento para fins de validação [52–54], compartilhamento e reutilização de conhecimento, verificação de qualidade de dados e atribuição de

resultados científicos. Um dos conceitos comumente capturados na proveniência é o de *causalidade*, que é dado pelas relações de dependência existentes entre processos e conjuntos de dados. Estas dependências podem derivar, por transitividade, dependências entre conjuntos de dados e entre processos. De acordo com Davidson et al. [34], causalidade pode ser inferida tanto da proveniência prospectiva como da retrospectiva. Detalhes adicionais relevantes sobre cada conjunto de dados, como descritores do domínio científico, ou sobre os processos, como detalhes de tempo de execução, também podem ser capturados. O processo de captura pode ocorrer de forma automatizada, através de agentes de coleta, ou de forma manual, através de anotações feitas por cientistas. Diversos sistemas de proveniência foram implementados [55–65], geralmente acoplados a SGWCs. Informações de proveniência têm diversas aplicações e tradicionalmente são utilizadas para análise *a posteriori* de experimentos científicos computacionais, como a exploração de dados resultantes dos mesmos. A interoperabilidade entre estes sistemas foi avaliada durante a série de eventos *Provenance Challenge* [66, 67], a partir dos quais foi proposto um modelo de proveniência padronizado, o OPM (*Open Provenance Model*) [51], que permite a troca de informações de proveniência.

No contexto de computação paralela e distribuída, a gerência de proveniência apresenta diversos desafios [59, 60, 63, 68–70]. O modelo utilizado para representar a proveniência deve prever eventos típicos de sistemas distribuídos, como falhas de rede e de recursos computacionais e tentativas de ataque por usuários maliciosos. Em função de falhas, uma chamada a uma atividade de um workflow pode ter diversas tentativas de execução, logo múltiplas ocorrências dessas atividades podem aparecer nos registros de proveniência. A própria captura das informações de proveniência é dificultada pela heterogeneidade de tais sistemas, onde usuários remotos muitas vezes têm acesso restrito aos recursos computacionais, tendo somente as permissões necessárias para executar tarefas computacionais e armazenar dados de entrada e saída. Miles et al. [71] enumeram diversos requisitos para a gerência de proveniência em infraestruturas de computação distribuída, como a integração de dados de repositórios de proveniência distintos, a preservação da integridade das informações de proveniência e a escalabilidade. Sistemas distribuídos que se utilizam de uma rede aberta como a Internet estão naturalmente sujeitos a ataques realizados por usuários maliciosos. Algumas questões relevantes em segurança de proveniência são ligadas à atribuição correta de dados científicos [44, 72] e do experimento científico como um todo [73]; e à privacidade dos dados de proveniência [74], que em áreas como a medicina pode ser crítica por envolver dados sobre a saúde de indivíduos.

O acesso à proveniência é uma das questões importantes na análise de experimentos científicos computacionais pois é por meio do qual que se pode explorar os seus resultados através, por exemplo, da verificação dos parâmetros científicos

dados como entrada e gerados no seu decorrer, do encadeamento ocorrido entre as atividades pela produção e consumo de conjuntos de dados e da cronologia de eventos. Cientistas também podem consultar informações de proveniência para buscar especificações de workflows científicos que já tenham sido utilizadas para resolver algum problema em questão, ajustando-as e reutilizando-as conforme variações de parâmetros e atividades que sejam de interesse. Simmhan et al. [44] observam que a proveniência é usualmente acessada visualmente em grafos de proveniência [75], através de linguagens de consulta, como SQL [64, 76] e SPARQL [62, 77], ou através de interfaces de programação de aplicações [78]. Algumas abordagens utilizam linguagens declarativas como Prolog [65, 79] e Datalog [80] para consultas e deduções automatizadas, características do paradigma de programação lógico. Linguagens voltadas especificamente para a consulta de informações de proveniência [81–86] buscam facilitar a travessia de grafos de proveniência, derivados das relações de produção de consumo entre atividades e conjuntos de dados, e consultas sobre conjuntos hierárquicos de dados. Durante o *Third Provenance Challenge*, detectamos [64] problemas existentes nas soluções existentes para gerência de proveniência:

- A ausência de informações do domínio científico dificultava a análise e interpretação dos dados de proveniência.
- A escrita de consultas em linguagens genéricas como SQL, XQuery/XPath e SPARQL era demasiadamente trabalhosa pois envolvia, por exemplo, uso demasiado de junções e cálculos frequentes do fecho transitivo das relações de derivação de dados.

Tais restrições limitavam tanto a aplicabilidade e utilidade das informações de proveniência, assim como a usabilidade dos sistemas de gerência de proveniência por cientistas com pouca experiência em bancos de dados.

1.2 Definição do Problema

Neste trabalho, abordamos o problema da gerência de informações de proveniência sobre computações que envolvam muitas tarefas (*many-task computing* [87]), implementadas por meio de workflows científicos, e que sejam realizadas em ambientes paralelos e distribuídos de alto desempenho. Neste contexto de larga escala, as questões envolvidas na gerência de proveniência tornam-se mais complexas pois os mecanismos utilizados na coleta e armazenamento dessas informações podem ter um impacto negativo na escalabilidade do experimento computacional se, por exemplo, o volume das mesmas for demasiadamente grande. Por outro lado, o nível de detalhe das informações capturadas deve ser suficiente para permitir a análise do experimento de maneira satisfatória. Além de permitir a análise do processo de derivação

de dados de um experimento, a proveniência pode apoiar a depuração e otimização de estratégias de execução e de manuseio de dados de workflows, se as respectivas informações relativas à utilização de recursos computacionais, como memória, armazenamento e tempo de processamento, forem adicionadas. Isso é importante para experimentos em larga escala, onde o tempo disponível para utilização dos recursos computacionais é concorrido e limitado. Após a coleta e armazenamento das informações de proveniência, nem sempre o cientista dispõe de mecanismos de fácil uso para acessá-las. Assim, interfaces que permitam a elaboração e execução relativamente simples de consultas sobre as informações de proveniência também são importantes no apoio à análise de um experimento computacional.

O objetivo desta tese é prover uma ferramenta escalável de gerência de proveniência de workflows paralelos e distribuídos que leve em consideração os seguintes aspectos:

- *Modelagem.* Determinar um modelo de proveniência que seja adequado para capturar a especificação do experimento (*proveniência prospectiva*), assim como eventos que ocorram durante a sua realização (*proveniência retrospectiva*). O modelo deve levar em consideração questões importantes em computação paralela e distribuída como consumo de recursos computacionais, tolerância a falhas e segurança, permitindo o registro de eventos relacionados.
- *Utilidade.* Explorar como as informações de proveniência podem contribuir para uma melhor gerência das diversas fases de um experimento científico computacional, tanto do ponto de vista da interpretação dos resultados, através da coleta de informações relevantes do domínio científico geradas no decorrer de um experimento, como da utilização eficiente dos recursos computacionais paralelos e distribuídos.
- *Usabilidade.* Prover um mecanismo de acesso às informações de proveniência que permita que consultas realizadas com frequência pelos usuários sejam expressas de forma relativamente simples.
- *Segurança.* Determinar que controles de segurança podem ser adotados para proteger as informações de proveniência. Novos resultados científicos podem ser derivados a partir da análise de um experimento, o que pode produzir propriedade intelectual de valor. Portanto, a facilidade de reprodutibilidade de experimentos, oferecidas pelos sistemas de proveniência, também pode ser vista como uma ameaça à propriedade intelectual, se os mecanismos de segurança adequados não forem adotados para proteger informações de proveniência.

Os resultados desta tese foram publicados em anais de eventos, periódicos e relatórios técnicos. Em particular, a seguinte seleção de artigos contém as principais

contribuições realizadas:

- L. Gadelha, M. Mattoso. Kairos: An Architecture for Securing Authorship and Temporal Information of Provenance Data in Grid-Enabled Workflow Management Systems. *Proceedings of the 4th IEEE International Conference on e-Science (e-Science 2008)*, p. 597-602. IEEE Computer Society, 2008.
- L. Gadelha, B. Clifford, M. Mattoso, M. Wilde, and I. Foster. Provenance Management in Swift. *Future Generation Computer Systems*, 27(6):775-780. Elsevier, 2011.
- L. Gadelha, M. Mattoso, M. Wilde, I. Foster. Provenance Query Patterns for Many-Task Scientific Computing. *3rd USENIX Workshop on the Theory and Practice of Provenance (TaPP 2011)*, USENIX, 2011.
- L. Gadelha, M. Wilde, M. Mattoso, I. Foster. MTCProv: A Practical provenance query framework for many-task scientific computing. *Distributed and Parallel Databases*, 30(5-6):351-370. Springer, 2012.

Em junho de 2009 participamos do evento *Third Provenance Challenge* em Amsterdã (Holanda), integrando a equipe do Swift [88], um SGWC para ambientes paralelos e distribuídos. O objetivo do evento era avaliar a interoperabilidade entre sistemas de gerência de proveniência. As funcionalidades de gerência de proveniência do Swift foram demonstradas e avaliadas, o que deu origem à maior parte das linhas de trabalho desta tese com vistas a melhorar a aplicabilidade e usabilidade do sistema, sem prejudicar a escalabilidade dos workflows executados.

1.3 Organização da Tese

No capítulo 2, apresentamos conceitos importantes para mais fácil leitura dos capítulos subsequentes. No capítulo 3, apresentamos uma avaliação comparativa com trabalhos existentes. No capítulo 4, apresentamos a metodologia adotada e os resultados desta tese em linhas gerais. Os detalhes teóricos e de desenvolvimento são apresentados nos apêndices da tese, conforme o problema abordado. Os apêndices são baseados em trabalhos publicados em relatórios técnicos, anais de eventos e periódicos [64, 73, 89–94] durante o desenvolvimento da tese. No apêndice A, é apresentado um modelo de proveniência [64] que segue o padrão vigente, o OPM (*Open Provenance Model*) [51], para capturar relações de consumo e produção entre dados e atividades de um workflow científico. O modelo estende o OPM para registrar também eventos típicos de ambientes paralelos e distribuídos, como uma chamada a uma atividade possivelmente ter várias tentativas de execução devido a

falhas. O apêndice também descreve uma implementação [91] baseada neste modelo que foi incorporada ao Swift [88, 95, 96], um SGWC para ambientes paralelos e distribuídos, e avaliada durante o *Third Provenance Challenge* [67]. No apêndice B, iniciamos fazendo um levantamento das consultas frequentemente utilizadas para analisar informações de proveniência [92] de computações compostas por muitas tarefas. Estas consultas são classificadas em padrões e discutimos como os mesmos podem ser suportados por sistemas de gerência de proveniência. Em seguida, apresentamos o MTCProv [94], um sistema de gerência de proveniência para computações dadas por muitas tarefas, que consiste de modificações realizadas no nosso modelo de proveniência para melhor suporte aos padrões de consulta levantados. Ele é composto também por uma interface de acesso a informações de proveniência que permite a visualização de grafos de proveniência e a realização de consultas sobre informações de proveniência usando uma sintaxe simplificada em relação ao SQL. A aplicação OOPS (*Open Protein Simulator*) [97], de modelagem da estrutura tridimensional de proteínas, é utilizada na avaliação do MTCProv. No apêndice C, estudamos os aspectos de segurança envolvidos na gerência de informações de proveniência [90]. Apresentamos uma arquitetura de segurança, chamada Kairos [73], que permite a proteção de autoria de experimentos científicos computacionais executados em ambientes distribuídos.

Capítulo 2

Conceitos Preliminares

Neste capítulo são apresentados os principais conceitos utilizados nos demais capítulos da tese.

2.1 Experimentos Científicos Computacionais

Experimentos científicos computacionais geralmente são dados pela composição de um grande número de atividades, que podem depender de dados produzidos por outras atividades. Uma *atividade* é um passo lógico dentro de um processo. Pode ser uma *atividade manual*, que exige intervenção humana, ou uma *atividade automatizada*. A gerência do ciclo de vida [2, 98–100] destes experimentos *in-silico*, ilustrado na figura 2.1, é também um aspecto importante da e-Ciência e consiste em gerenciar todas as etapas que ocorrem durante a vida do experimento, elas podem ser agrupadas em três fases principais:

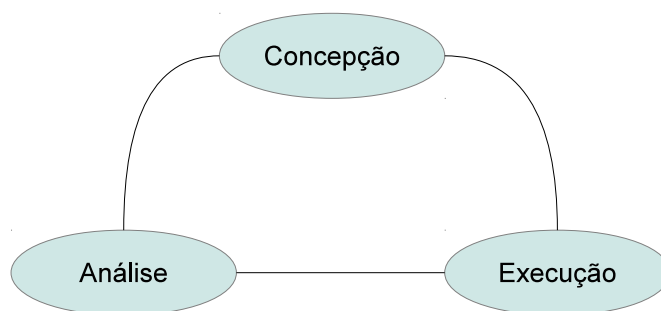


Figura 2.1: Ciclo de vida de um experimento científico computacional [2].

- uma fase de *composição*, a qual consiste em especificar as atividades que compõem o experimento, os tipos de dados de entrada e saída, e as dependências de dados existentes entre as atividades,
- uma fase de *execução* (ou *orquestração*), que consiste da realização das atividades definidas na fase de composição,

- uma fase de *análise*, que consiste em estudar os dados gerados pela fase de execução.

Estes experimentos computacionais frequentemente são compostos por um grande número de tarefas computacionais, onde algumas destas tarefas podem depender de dados de saída gerados por outras tarefas. Uma vez especificado o experimento, é necessário submeter as suas atividades e transferir seus dados de entrada para os recursos computacionais disponíveis. Isso requer uma certa coordenação sobre o escalonamento de tarefas e transferências de dados devido às dependências de dados existentes. Após a execução do experimento, os cientistas têm que analisar os dados resultantes. Esta tarefa pode ser facilitada se o cientista tiver acesso a metadados contendo informações detalhadas sobre os resultados do experimento. O ciclo de especificar, executar e analisar um experimento computacional pode ser repetido a fim de fazer refinamentos até que o cientista esteja satisfeito com os resultados.

Sistemas que apoiam a gerência de experimentos científicos computacionais podem oferecer algumas das seguintes funcionalidades:

- *Abstração*. Permitir ao cientista especificar o experimento em alto nível, em termos de atividades e dependências de dados, sem precisar detalhar aplicações específicas e onde elas serão executadas.
- *Reprodutibilidade*. Permitir que o experimento computacional seja reproduzido por terceiros de forma independente e automatizada. Uma aplicação importante são as publicações científicas inteligentes [101], cujos resultados podem ser reproduzidos por leitores e revisores de forma automatizada.
- *Reutilização*. Ser possível reutilizar especificações de experimentos realizados previamente para concepção de novos experimentos. Uma iniciativa para facilitar a reutilização de experimentos é o myExperiment [102], um repositório de especificações de workflows científicos que pode ser acessado livremente.
- *Escalabilidade*. O experimento deve ser resiliente ao aumento do número de tarefas e da quantidade de dados.

A utilização de tais sistemas pode tornar a gerência do ciclo de vida de experimentos computacionais em larga escala uma tarefa tratável.

2.2 Workflows Científicos

A descrição completa ou parcial de uma experimento científico computacional em termos de suas atividades, controles de fluxo e dependências de dados é chamada de um *workflow científico* [35–38]. Será utilizado simplesmente o termo *workflow* para

representar o mesmo conceito. Também será utilizado o termo *aplicação componente* para descrever um programa de computador, que implemente a correspondente atividade de um experimento científico computacional. Conceitualmente, um workflow pode ser visto como um grafo dirigido onde os nós são as aplicações componentes e as arestas representam os fluxos de dados. Esse grafo pode ser acíclico, indicando que o workflow não contém controles de fluxo do tipo laço, onde um mesmo trecho do workflow poderia ser repetido até que alguma condição fosse satisfeita. Um *workflow abstrato* consiste em uma descrição de alto nível de um experimento computacional, sem detalhes do planejamento de sua execução. Um *workflow concreto* especifica os recursos computacionais que serão utilizados para a sua execução. Um experimento científico computacional pode ser composto por diversos workflows científicos, que seguem um ciclo de vida análogo ao dos experimentos científicos computacionais.

Um *sistema de gerência de workflows científicos* (SGWC) é uma ferramenta que permite a especificação, execução e análise de workflows científicos. Linguagens de coordenação, como Linda [103–105] e Strand [106, 107], podem ser consideradas precursoras dos sistemas de gerência de workflows científicos atuais. Estas linguagens provêm mecanismos para composição paralela de programas sequenciais. Alguns dos aspectos envolvidos no projeto de SGWCs incluem monitoramento de execução [89], reuso de software e gerência de configuração [108], escalonamento e execução [109–111] e segurança [73].

A especificação é normalmente realizada utilizando-se uma *linguagem de workflow*, que permite a especificação das interações existentes entre as atividades de um workflow científico, ilustrada na listagem 2.1 com um workflow de renderização de imagens, ou através de uma interface gráfica que permita a cientistas sem conhecimentos avançados de informática especificar workflows de forma mais fácil através da seleção de componentes pré-construídos, ilustrada na figura 2.2.

Listagem 2.1: Composição textual de workflow no Swift.

```

1 type scene;
2 type image;
3 type scene_template;
4
5 int threads;
6 int steps;
7 string resolution;
8
9 threads = @toint(@arg("threads","8"));
10 resolution = @arg("resolution","1920x1080");
11 steps = @toint(@arg("steps","10"));
12 scene_template template <"tscene">;
13
14 app (scene out) generate (scene_template t, int i, int total)
15 {
16     genscenes i total @filename(t) stdout=@out;
17 }
18
```

```

19 app (image o) render (scene i, string r, int t)
20 {
21     cray "-s" r "-t" t stdin=@i stdout=@o;
22 }
23
24 scene[] scene_files <simple_mapper; location=".", prefix="scene.", suffix=".data">;
25
26 image[] image_files <simple_mapper; location=".", prefix="scene.", suffix=".ppm">;
27
28 foreach i in [1:steps] {
29     scene_files [i] = generate(template, i, steps);
30     image_files [i] = render(scene_files [i], resolution, threads);
31 }

```

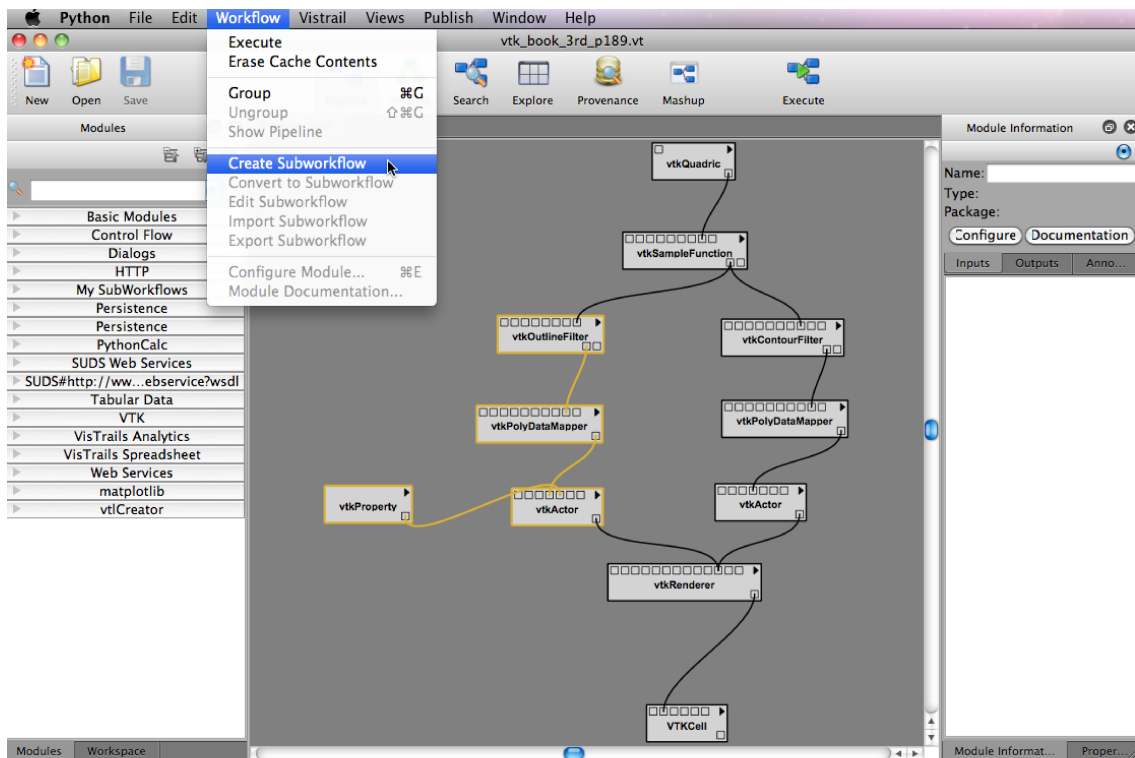


Figura 2.2: Composição gráfica de workflow: Vistrails [3]. Fonte: <http://www.vistrails.org>.

Diversas implementações de SGWCs encontram-se disponíveis, como por exemplo:

- Taverna [112], onde as aplicações componentes geralmente são serviços web, especialmente de bioinformática.
- Pegasus [113], voltado para a execução distribuída de workflows estruturados como grafos acíclicos dirigidos em grades computacionais.
- Kepler [114], cujo modelo de execução é baseado em *atores* [115] e a composição do workflow é gráfica.

- Vistrails [3], registra a evolução da especificação do workflow e é utilizado principalmente na área de visualização científica.
- Askalon [116], executa as atividades componentes de um workflow em grades computacionais.
- Chiron [111], utiliza uma abordagem algébrica semelhante ao modelo relacional para modelagem e otimização de execução paralela de workflows.
- Swift [88], suporta paralelismo implícito e possui modelo de execução que permite a utilização de computadores paralelos e grades computacionais.

van der Aalst et al. [117–119] agruparam workflows conforme cerca de 40 padrões de controle de fluxo utilizados, como os padrões de sequência, bifurcação paralela, sincronização, e laço. As figuras 2.3, 2.4, 2.5 e 2.6 ilustram esses respectivos padrões de controle de fluxo, onde um retângulo representa uma atividade, e uma elipse representa um conjunto de dados. A identificação desses padrões permite avaliar a expressividade dos diversos SGWCs e auxilia o cientista na escolha de quais destes seriam mais adequados ao seu experimento científico computacional.



Figura 2.3: Padrão de controle de fluxo: sequência.

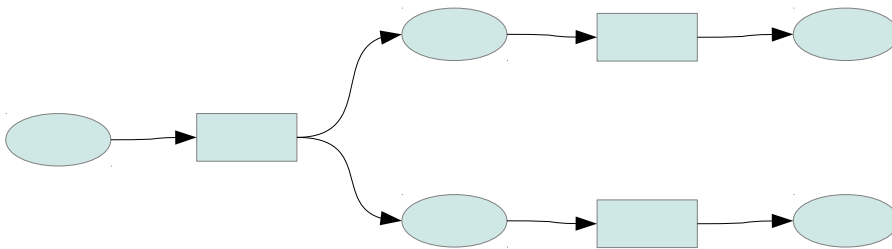


Figura 2.4: Padrão de controle de fluxo: bifurcação paralela.

Uma *máquina de execução* planeja a execução das atividades do workflow localmente ou em recursos computacionais remotos, como clusters e ambientes de computação distribuída. As atividades abstratas são mapeadas para aplicações concretas disponíveis nos recursos computacionais existentes. O escalonamento da execução das aplicações pode ser realizado utilizando-se tanto mecanismos internos ao SGWC, assim como mecanismos externos, como gerenciadores de filas de *clusters*.

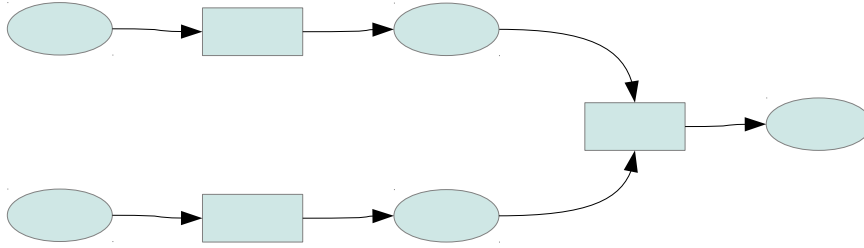


Figura 2.5: Padrão de controle de fluxo: sincronização.

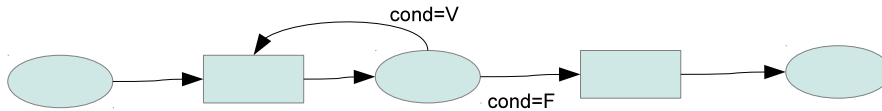


Figura 2.6: Padrão de controle de fluxo: laço.

Computações em larga escala em ambientes distribuídos são naturalmente sujeitas a falhas. Assim, é interessante que um SGWC possua mecanismo para detecção de falhas de execução de tarefas e de respectiva recuperação, permitindo a resubmissão de tarefas quando necessário. SGWCs com suporte a infraestruturas de computação em grade podem tirar proveito da escalabilidade oferecida por esses ambientes.

Yu e Buyya [120] realizaram um levantamento de SGWCs que suportam computação em grade e desenvolveram uma taxonomia que foi usada para classificá-los de acordo com as seguintes características: como o workflow é concebido, como as suas tarefas são escalonadas (execução), quais são seus mecanismos de tolerância a falhas, e como a transferência de dados é realizada. O trabalho identificou várias soluções utilizadas na implementação de cada um destes sistemas, e apontou quais os desafios que ainda deveriam ser abordados. Muitos pesquisadores, especialmente nas ciências da vida, não confiam em executar seus workflows em ambientes remotos via rede, uma vez que têm preocupações de propriedade intelectual [121]. Consequentemente, a segurança é também um aspecto relevante da gestão de workflows científicos em ambientes distribuídos.

SGWCs permitem aos cientistas automatizar a maior parte da especificação e execução de um experimento computacional científico. No entanto, eles precisam acessar informações que descrevam o que aconteceu no experimento, a fim de analisar seus resultados. Para apoiar a análise do workflow científico, metadados relacionados ao domínio científico e informações de proveniência podem ser coletados. Essas informações podem descrever mudanças e a evolução da especificação do workflow, assim como o consumo e produção de dados por aplicações componentes executadas.

2.3 Proveniência

Informações de *proveniência* [34, 42–44, 46, 47, 49–51] descrevem as relações de consumo e produção existentes entre conjuntos de dados e aplicações componentes de um workflow, permitindo a análise do histórico de derivação de dados de um experimento computacional. Quais usuários iniciaram ou interromperam aplicações componentes, qual foi a especificação do workflow utilizada, metadados do domínio científico, e estatísticas do ambiente de execução das aplicações componentes também são informações que podem fazer parte da proveniência de um experimento computacional. Simmhan et al. [44] classificam sistemas de gerência de proveniência de acordo com:

- *Aplicação*: a finalidade e utilização das informações de proveniência, como reprodutibilidade [101], avaliação da qualidade de dados, trilhas de auditoria e proteção de propriedade intelectual [73].
- *Conteúdo*: os aspectos da proveniência que são coletados, como histórico de derivação de dados (*proveniência retrospectiva*) ou a especificação do workflow (*proveniência prospectiva*);
- *Representação*: o modelo de dados utilizado para descrever a proveniência;
- *Armazenamento*: modo de armazenamento físico das informações, como bancos de dados relacionais ou arquivos texto;
- *Disseminação*: as formas de acesso às informações de proveniência, como linguagens de consulta.

O *Open Provenance Model* (OPM) [51] é um modelo para proveniência que busca permitir a interoperabilidade entre diferentes sistemas de gerência de proveniência. São definidos conceitos de *artefato*, *processo* e *agente*, que geralmente correspondem, no contexto de workflows, aos conceitos de conjunto de dados, atividade componente e usuário. Um *grafo de proveniência* é um grafo dirigido onde os nós são artefatos, processos ou agentes, e as arestas pertencem a uma das categorias apresentadas na figura 2.7. Uma aresta representa uma dependência causal entre a origem e o destino. Na figura 2.7, são definidos os relacionamentos causais: **usou** (artefato foi usado por um processo), **foiGeradoPor** (artefato foi gerado por processo), **foiIniciadoPor** (processo iniciado por processo), **foiDerivadoDe** (artefato derivado de artefato) e **foiControladoPor** (processo controlado por agente). Em relacionamentos com multiplicidade n para 1, as arestas são distinguidas pelo *papel* que representam no relacionamento, denotado por (R) na figura 2.7. Um *papel* designa a função que um artefato ou um agente tiveram em um processo. Dois grafos de proveniência podem

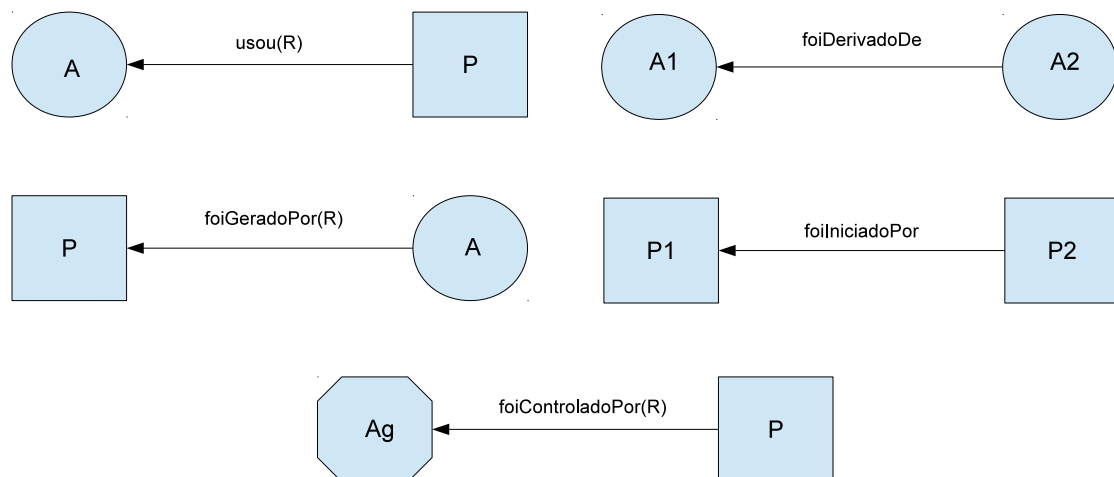


Figura 2.7: Arestas do OPM.

descrever um mesmo processo de derivação de dados de formas diferentes. Na figura 2.8, os grafos apresentam duas *descrições* distintas de uma mesma execução.

Mais recentemente, o OPM evoluiu para o PROV [122], um modelo de dados para proveniência proposto pelo W3C (*World Wide Web Consortium*).

2.4 Swift

Nesta tese, o trabalho de implementação e os experimentos foram realizados com o Swift [88], um SGWC composto por uma linguagem de alto nível para especificação de workflows científicos como scripts, ilustrada na figura 2.1, e um ambiente de execução com suporte nativo a execução local, em clusters e distribuída. O componente atual de gerência de proveniência [64, 94] do Swift foi projetado e implementado no decorrer desta tese.

O Swift [88, 95, 96] é um SGWC para composição de várias tarefas computacionais (*many-task computing*) em ambientes paralelos e distribuídos. Tais tarefas podem ser fracamente acopladas (*bag of tasks*), ou fortemente acopladas, tanto locais (paralelismo com memória compartilhada) ou distribuídas (paralelismo com memória distribuída).

A linguagem de especificação de workflow do Swift suporta paralelismo implícito, ou seja, toda expressão de um script Swift é avaliada em paralelo, exceto quando há dependências de dados. Isso se aplica também ao processamento de coleções de dados com o controle de fluxo `foreach`, que executa instruções em paralelo para cada elemento da coleção. Desta forma, não é possível assumir uma ordem de execução dessas expressões. A forma como a especificação do workflow é separada da configuração do ambiente de execução permite independência de localidade, de

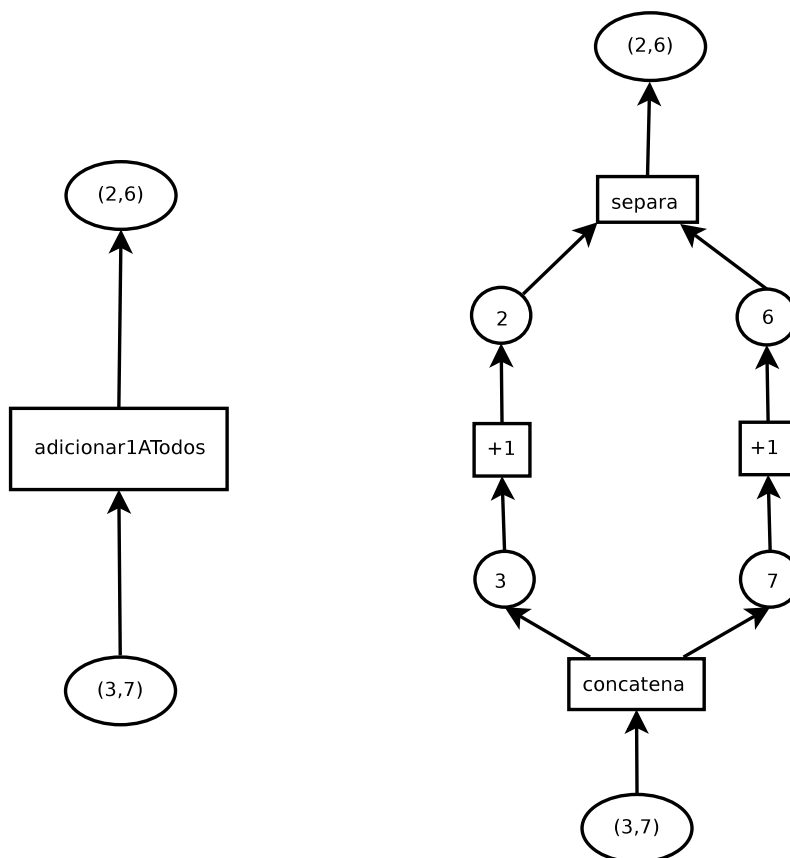


Figura 2.8: Descrições distintas de uma mesma execução. Fonte: Moreau et al. [4]

modo que os scripts não expressam explicitamente transferências de dados e seleção de sítios de execução. A

Aplicações que compõem um workflow são associadas a funções (*app functions*) em um script Swift, e as variáveis podem ser associadas a: tipos de dados primitivos, dados persistentes armazenados em arquivos e tipos de dados compostos (coleções).

A linguagem possui uma sintaxe semelhante ao C e ao Java. Em geral a estrutura de um script Swift é dada por declarações de tipos de dados, variáveis e funções (de chamada a aplicações ou compostas). Em seguida podem ser encontradas atribuições de variáveis, chamadas a funções e controles de fluxo de execução.

O Swift possui algumas semelhanças com paradigma de programação funcional como computação orientada a avaliação de funções, chamadas a funções sem *efeitos colaterais*, avaliação tardia de expressões e variáveis de atribuição única. A listagem 2.2 ilustra um workflow para paralelização do BLAST [123], um procedimento para alinhamento de sequências biológicas. Nas linhas 1 a 21, são definidos tipos de dados e mapeamentos para arquivos em disco. Nas linhas 23 a 43, são definidas as funções para chamada de aplicações componentes do workflow: uma função para particionamento do banco de dados de sequências `split_database`, uma função para indexação e formatação de bancos de dados de sequências `formatdb`, uma função

para realização do alinhamento de sequências `blastall` e uma função para fusão de bancos de dados de sequências. Na linha 45 é chamada a função que particiona o banco de dados de entrada. Nas linhas 47 e 48 são definidos mapeadores para arquivos em disco que receberão os resultados dos alinhamentos. Nas linhas 50 a 54 são realizados os alinhamentos entre a sequência de entrada e cada uma das partições do banco de dados de sequências de entrada. O laço `foreach` é intrinsecamente paralelo, alocando cada alinhamento para um recurso computacional que esteja disponível. Finalmente, na linha 56, é realizada a fusão dos resultados dos alinhamentos.

O motor de execução do Swift paraleliza automaticamente as execuções de aplicações componentes que não tenham dependências mútuas de dados. Uma heurística de balanceamento de carga distribui de forma adaptativa as execuções das aplicações componentes entre os recursos disponíveis. A heurística associa uma pontuação de desempenho para cada recurso computacional, que se ajusta à taxa de execução de tarefas alcançada pelo mesmo. Técnicas de tolerância a falhas re-submetem automaticamente execuções que falharam sem repetição desnecessária de computações. Submissões redundantes podem ser utilizadas para tentar diminuir a espera em filas de escalonamento de clusters. O motor de execução também é responsável pela submissão de aplicações componentes e transferência dos arquivos requeridos. Diversos provedores de dados e de execução dão suporte a diferentes ambientes de execução, como execução local, em clusters e em grid. Um catálogo de recursos computacionais, ilustrado na listagem 2.3, lista os recursos disponíveis e a configuração dos mesmos. Um catálogo de aplicações componentes, ilustrado na listagem 2.4, lista em que recursos computacionais as aplicações componentes estão disponíveis. A segunda coluna contém o identificador da aplicação componente, que é utilizado na especificação do workflow, listagem 2.2, nas linhas 25, 30, 36 e 42. Para cada chamada a uma aplicação componente, o motor de execução realiza os seguintes passos:

- verifica no catálogo de aplicações em quais recursos computacionais a aplicação está disponível,
- seleciona um recurso computacional (maior probabilidade para recurso com maior pontuação de desempenho),
- verifica no catálogo de recursos computacionais que provedor de execução e provedor de dados deve ser utilizado,
- se for o caso, transfere os dados para o recurso computacional,
- executa a aplicação componente,

Listagem 2.2: Paralelização do BLAST no Swift.

```
1 type fastaseq;
2 type headerfile;
3 type indexfile;
4 type seqfile;
5 type database
6 {
7     headerfile phr;
8     indexfile pin;
9     seqfile psq;
10 }
11 type query;
12 type output;
13 string num_partitions=@arg("n", "8");
14 string program_name=@arg("p", "blastp");
15 fastaseq dbin <single_file_mapper;file=@arg("d", "database")>;
16 query query_file <single_file_mapper;file=@arg("i", "sequence.seq")>;
17 string expectation_value=@arg("e", "0.1");
18 output blast_output_file <single_file_mapper;file=@arg("o",
19                                     "output.html")>;
20 string filter_query_sequence=@arg("F", "F");
21 fastaseq partition[] <ext;exec="splitmapper.sh",n=num_partitions>;
22
23 app (fastaseq out[]) split_database (fastaseq d, string n)
24 {
25     fastasplitn @filename(d) n;
26 }
27
28 app (database out) formatdb (fastaseq i)
29 {
30     formatdb "-i" @filename(i);
31 }
32
33 app (output o) blastapp(query i, fastaseq d, string p, string e, string f,
34                        database db)
35 {
36     blastall "-p" p "-i" @filename(i) "-d" @filename(d) "-o" @filename(o)
37             "-e" e "-T" "-F" f;
38 }
39
40 app (output o) blastmerge(output o_fragments[])
41 {
42     blastmerge @filename(o) @filenames(o_fragments);
43 }
44
45 partition=split_database(dbin, num_partitions);
46
47 database formatdbout[] <ext; exec="formatdbmapper.sh",n=num_partitions>;
48 output out[] <ext; exec="outputmapper.sh",n=num_partitions>;
49
50 foreach part,i in partition {
51     formatdbout[i] = formatdb(part);
52     out[i]=blastapp(query_file, part, program_name, expectation_value,
53                   filter_query_sequence, formatdbout[i]);
54 }
55
56 blast_output_file=blastmerge(out);
```

Listagem 2.3: Catálogo de recursos computacionais.

```
1 <config>
2   <pool handle="localhost" sysinfo="INTEL32::LINUX">
3     <gridftp url="local://localhost" />
4     <execution provider="local" url="none" />
5     <workdirectory>/prj/prjssi/lgadelha/swiftwork</workdirectory>
6     <profile namespace="karajan" key="jobThrottle">0.03</profile>
7     <profile namespace="swift" key="stagingMethod">file</profile>
8   </pool>
9
10  <pool handle="sunhpc.lncc.br">
11    <gridftp url="local://localhost" />
12    <execution jobmanager="local:sge" provider="coaster"/>
13    <profile key="jobsPerNode" namespace="globus">8</profile>
14    <profile key="slots" namespace="globus">128</profile>
15    <profile key="nodeGranularity" namespace="globus">1</profile>
16    <profile key="maxNodes" namespace="globus">16</profile>
17    <profile namespace="globus" key="maxtime">36000</profile>
18    <profile namespace="globus" key="pe">mpi</profile>
19    <profile namespace="globus" key="queue">linux.q</profile>
20    <profile key="jobThrottle" namespace="karajan">1.27</profile>
21    <profile namespace="karajan" key="initialScore">10000</profile>
22    <filesystem provider="local" url="none" />
23    <workdirectory>/prj/prjssi/lgadelha/swiftwork</workdirectory>
24  </pool>
25 </config>
```

Listagem 2.4: Catálogo de aplicações componentes.

1	localhost	fastasplitn	/prj/prjssi/lgadelha/parallelblast/fastasplitn
2	sunhpc.lncc.br	formatdb	/hpc/blast/2.2.22/bin/formatdb
3	sunhpc.lncc.br	blastall	/hpc/blast/2.2.22/bin/blastall
4	localhost	blastmerge	/prj/prjssi/lgadelha/parallelblast/blastmerge

- transfere de volta os dados resultantes.

Tanto para as transferências de dados quanto para a submissão de tarefas diversos provedores associados a diferentes implementações podem ser utilizados. As transferências podem ser gerenciadas por provedores específicos para, por exemplo, SSH ou GridFTP [23]. A submissão pode ser gerenciada por provedores para escalonadores como o PBS [124] e o GRAM [125].

2.5 Segurança da Informação

O objetivo da segurança da informação é identificar as ameaças, os riscos e os danos potenciais para ativos de informação, e propor mecanismos para reduzir ou eliminar o risco de estas ameaças serem exploradas. Os primeiros sistemas de computação normalmente foram instalados em ambientes isolados física e logicamente em grandes corporações e centros de pesquisa. Portanto, as ameaças a esses sistemas eram relativamente fáceis de enumerar, uma vez que o número de pessoas que tinham acesso a eles era relativamente pequeno e bem conhecido. Com o desenvolvimento de sistemas distribuídos e da Internet, sistemas de computação tornaram-se muito mais expostos, sendo consideravelmente mais difícil enumerar as ameaças existentes, uma vez que usuários mal-intencionados podem obter acesso não-autorizado aos mesmos explorando falhas na arquitetura do sistema ou aplicação. Esta tese segue terminologia de segurança da informação utilizada por Anderson [126]. Uma *entidade* pode ser definida como uma pessoa, um sistema de computação ou uma organização. Entidades podem assumir diferentes funções em um sistema, que são chamadas de *papéis*. Uma *identidade* é definida como o nome de uma entidade. *Sigilo* pode ser definido como a propriedade alcançada quando o acesso a alguma informação é limitada a um número de entidades. Casos particulares de sigilo são a *confidencialidade*, quando um grupo de entidades pode limitar o acesso a algumas informações que elas compartilham, e a *privacidade*, quando uma entidade é capaz de limitar o acesso a alguma informação que ele possui. A *integridade* é a propriedade obtida quando é possível evitar que modificações não autorizadas sejam realizadas em uma informação. *Autenticidade* é a garantia da identidade de uma entidade em uma comunicação. Uma *ameaça* é um possível evento que pode comprometer a proteção de um sistema computacional. Uma *vulnerabilidade* é uma propriedade de um sistema que, em conjunto com uma ameaça, pode causar um sistema a ser comprometido. Um *adversário* pode ser definido como uma entidade que procura explorar alguma vulnerabilidade.

A abordagem de segurança proposta nesta tese utiliza técnicas de *infraestruturas de chaves públicas* (PKI, do inglês *public key infrastructure*) [127], onde existe

uma entidade confiável denominada *autoridade certificadora* (CA, do inglês *certificate authority*) que emite *certificados digitais* para outras entidades. Os certificados digitais são documentos eletrônicos que contêm informações, como a chave pública e o nome do dono da chave. O par formado pelo certificado digital e a chave privada correspondente é também chamado de *credencial*. Criptografia com a chave privada pode ser usada para realizar *assinaturas digitais*, algo que só o dono da chave privada pode fazer e que qualquer um pode verificar usando a chave pública correspondente. Os certificados digitais são assinados digitalmente pela CA para que outras entidades possam verificá-los. Em uma PKI há uma série de procedimentos que uma CA executa para verificar de modo seguro que uma entidade possui uma chave antes de emitir seu certificado digital correspondente. Técnicas de PKI são amplamente utilizados em diversas áreas da informática como ferramenta para a implementação de procedimentos de autenticação e para fornecer propriedades de segurança importantes, como a integridade e confidencialidade.

A GSI (*Grid Security Infrastructure*) [128, 129], amplamente utilizada em ambientes de rede, é fortemente baseada em conceitos de PKI. Os certificados digitais, emitidos aos usuários e recursos computacionais, são usados para autenticação cliente-servidor e para *login* único. Portanto, em ambientes de grade baseados na GSI pode-se supor que os usuários já possuam credenciais digitais e que sejam capazes de estabelecer as propriedades de segurança de integridade, confidencialidade e autenticidade.

A fim de determinar de forma segura a data e a hora em que um conjunto de dados foi gerado, uma técnica particularmente útil de PKIs é o protocolo de datação criptográfica (TSP, do inglês *time-stamp protocol*) [130–132]. O protocolo exige a existência de uma autoridade de datação criptográfica (TSA, do inglês *time-stamping authority*), que tem o seu próprio certificado digital. Para utilizar o TSP, um usuário calcula o valor de *hash* de um documento e o envia dentro de uma requisição de datação criptográfica para a TSA. A TSA gera um *token* que contém o valor de hash recebido e o tempo obtido a partir de uma fonte de hora de confiança. Este *token* é assinado digitalmente pela TSA e retornado ao usuário. O uso deste protocolo é ilustrado na figura 2.9. O usuário pode posteriormente usar o *token* assinado digitalmente como prova de que o documento existia em um determinado estado na hora contida no token. A confiabilidade de uma TSA envolve diversos aspectos, tais como os procedimentos de segurança e operacionais empregados, e a precisão e granularidade da fonte de tempo.

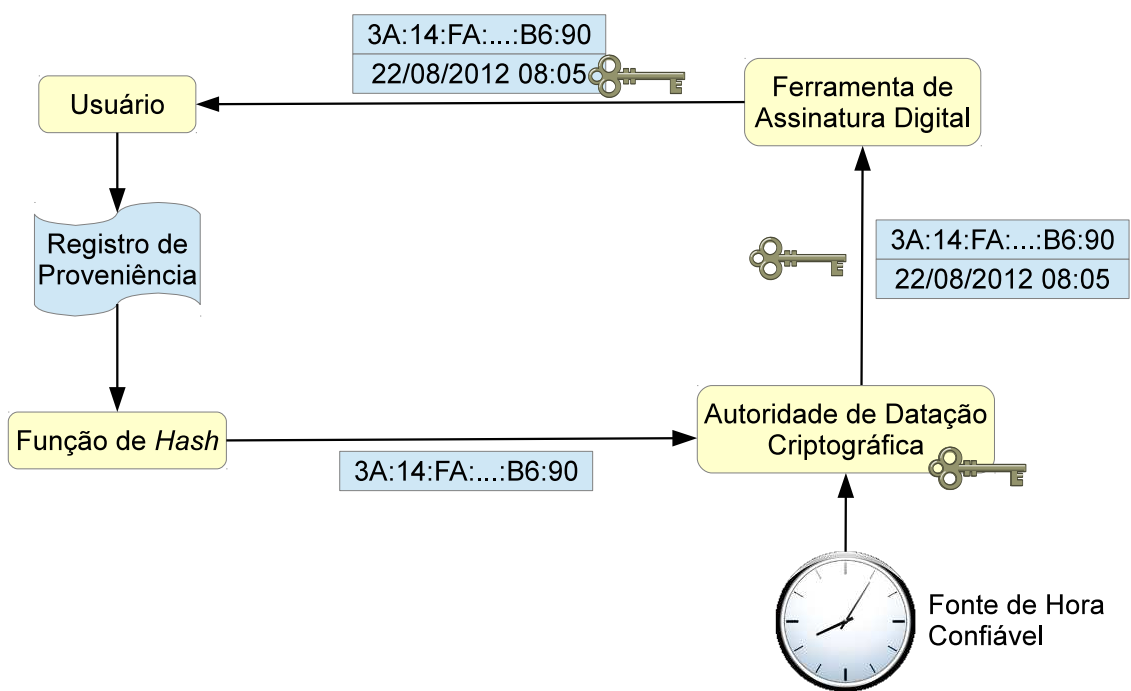


Figura 2.9: Protocolo de datação criptográfica TSP.

Capítulo 3

Trabalhos Relacionados

Como observado por alguns levantamentos sobre proveniência [44, 49], sistemas de proveniência variam conforme o conteúdo da informação de proveniência gravada, o seu nível de granularidade, e como ela é coletada, armazenada e consultada. Isto também foi observado durante o PC3, onde uma variedade de abordagens foi utilizada para realizar as atividades de gerência de proveniência propostas. Estes sistemas de proveniência usam uma variedade de modelos de dados, incluindo o semântico, o relacional e o semi-estruturado. No entanto, a maioria deles foi capaz de mapear seus modelos de dados para o OPM. Nesta seção, nos baseamos nos levantamentos sobre proveniência e na experiência do PC3 para comparar o MTCProv aos demais sistemas de gerência de proveniência.

3.1 Modelos de Proveniência

Alguns sistemas de proveniência não são dependentes de um determinado SGWC e funcionam como repositórios de proveniência acessíveis, por exemplo, como serviços web ou por meio de chamadas a funções de bibliotecas de programação. O Karma [133], por exemplo, é implementado como um serviço web e armazena informações de proveniência em um banco de dados relacional. Outro exemplo é o Tupelo [134], que é um *middleware* para gerência de dados e metadados científicos que utiliza técnicas da *web semântica*, tem uma biblioteca de programação para permitir a chamada a funções de armazenamento de informações de proveniência. Ao contrário do Swift, sistemas deste tipo muitas vezes exigem a instrumentação dos aplicativos que compõem um workflow científico, o que também foi observado durante o PC3. Isso pode ser difícil quando os usuários desses aplicativos não são também os seus desenvolvedores, ou se essas aplicações são dadas por código legado sem documentação.

Outra categoria é dada pelos SGWCs que têm suporte integrado à gerência de proveniência. O Vistrails [3], por exemplo, tem uma linguagem de consulta especializada em proveniência e utiliza bancos de dados relacionais e arquivos XML para

armazenar proveniência de dados, de processos e da evolução de workflows. Esta categoria também inclui o Swift, com o seu sistema integrado de gerência de proveniência MTCProv [94], que reúne informações tanto sobre os processos e dados envolvidos na execução de um *script* paralelo, assim como a sua própria especificação. Através do seu motor de execução, o Swift permite a execução em ambientes de computação paralela e distribuída de alto desempenho. Além disso, por ter um sistema integrado de proveniência, o Swift permite que seus usuários facilmente gerem e consultem registros de proveniência de seus experimentos. No Swift, a proveniência prospectiva [135] é dada pelo código fonte de um *script* Swift, que é armazenado no banco de dados de proveniência a cada execução do mesmo, registrando as diferentes versões utilizadas.

3.2 Análise de Tempo de Execução

Um exemplo da importância de se rastrear o consumo de recursos de rastreamento é o do TAIS (*Technology Audit and Insertion Service*) [136, 137], uma iniciativa para desenvolver ferramentas adequadas para monitoramento e auditoria de sistemas de computação de alto desempenho, que é um dos principais componentes de XSEDE [138], uma infraestrutura de computação distribuída de larga escala. A captura de informações em tempo de execução sobre o desempenho de workflows científicos paralelos e distribuídos pode ser útil no planejamento da execução das aplicações componentes. O Askalon [139] oferece um serviço de monitoramento de desempenho para workflows científicos que são executadas em ambientes de computação de grade. Um serviço de previsão de desempenho utiliza informações recolhidas pelo monitor de desempenho para apoiar um melhor planejamento de execução tarefas computacionais. Ogasawara et al. [111] armazenam o comportamento de uma execução paralela em bancos de dados de proveniência para melhorar o desempenho do motor de execução de workflows paralelos Chiron. O ParaTrac [140] reúne informações sobre o consumo de recursos por workflows científicos intensivos em dados, permitindo uma detalhada análise de desempenho, tais como a quantidade de dados que fluem entre as tarefas componentes. O PDB [141] coleta dados sobre tempo de execução de tarefas em workflows que utilizam *streaming*, ou seja, onde os fluxos de dados ocorrem em *buffers* de memória ao invés de arquivos, e também utiliza essas informações para o planejamento de execução. Nossa abordagem amplia a funcionalidade desses sistemas por coletar, além das informações de desempenho, as informações de proveniência e anotações sobre o domínio científico, permitindo a realização de consultas explorando estes diferentes aspectos.

3.3 Consultas

Linguagens específicas de domínio para a consulta de proveniência tem sido uma área de pesquisa ativa. O SGWC Vistrails [3] suporta experimentos científicos computacionais exploratórios, mantendo um registro da evolução da especificação de workflows científicos e das relações de derivação entre os dados e processos. Este modelo também permite que usuários criem anotações, na forma de pares chave-valor, vinculadas às entidades que compõem o registro de proveniência. A vtPQL [81] é uma linguagem específica de domínio para consultar informações de proveniência no Vistrails. É possível consultar a especificação do workflow, sua evolução e as relações de derivação entre os dados e processos. O sistema de consulta é acrescido de funções úteis no contexto de proveniência e workflows, tais como travessia de grafos de proveniência. Anand et al. [83] propõem uma representação de informações de proveniência baseada em relações de granularidade fina sobre conjuntos de dados aninhados. No mesmo trabalho é apresentada a linguagem de consultas para informações de proveniência QLP, que é independente de SGWC e do modelo de dados subjacente utilizado para armazenamento. A QLP é também fechada para a composição de consultas, e dispõe de funções para acessar versões específicas de conjuntos de dados aninhados produzidos por execuções de workflows. Os operadores da QLP agem como filtros sobre as relações de proveniência, retornando um subconjunto delas. Consultas de proveniência podem ser combinadas com consultas sobre estruturas de dados. QLP permite restringir a estrutura de caminhos sobre grafos de proveniência utilizando expressões regulares. A PQL [84] é utilizada para consulta de proveniência coletada em diferentes camadas, tais como a camada do workflow científico ou a do sistema operacional. Ela é derivada da linguagem de consulta para dados semi-estruturadas Lorel [142] e a estende para permitir a travessia bi-direcional de grafos e consultas que utilizam expressões regulares sobre as arestas do grafo para definir caminhos. Chebotko et al. [143] propõem o RDF-Prov, um sistema de gerência de proveniência que se baseia em técnicas da web semântica e armazenamento utilizando o modelo relacional. Ele gerencia ontologias de proveniência utilizando um motor de inferência, os dados de proveniência são representados como triplas RDF e SPARQL é utilizada para consultas.

A principal contribuição do MTCProv em relação às abordagens existentes é o seu suporte para captura, armazenamento e consulta de eventos que são típicos de computação paralela e distribuída de alto desempenho. Estes eventos incluem re-execuções para tolerância a falhas, submissões redundantes de tarefas para melhorar a taxa de execução de tarefas e o consumo de recursos em termos de operações de entrada e saída em sistemas de arquivos, utilização de memória e carga do processador. Outra contribuição do MTCProv, não encontrada em trabalhos relacionados,

foi demonstrar a sua escalabilidade em MTC, com mais de 1.000 núcleos de processamento no caso específico das aplicações utilizadas nos experimentos desta tese. Nosso trabalho pode ser distinguido em termos de sua contribuição para a utilidade e usabilidade de consultas de proveniência: a interface de consultas suporta consultas para padrões que são frequentemente utilizados pelos cientistas, mas que são difíceis de expressar com as atuais abordagens para consultas (como correlações entre informações científicas e de performance para diversas execuções de um mesmo workflow científico). Isto permite aos usuários expressar de forma relativamente fácil, com um nível modesto de conhecimento de SQL, consultas para acessar as informações de proveniência armazenadas em um banco de dados relacional que utiliza um esquema compatível com o OPM.

3.4 Segurança

Motivação para o estudo de questões de segurança de proveniência está presente em muitas pesquisas sobre proveniência. Em [71], Miles et al. identificam os requisitos ainda não atendidos por sistemas atuais para armazenamento e consulta de dados de proveniência. Dentre esses requisitos, eles mencionam *não-repúdio*, que é uma propriedade de segurança que protege contra litígios ilegítimos de autoria. Observam também que a informação temporal confiável pode ser importante para resolver disputas sobre direitos de propriedade intelectual. Embora alguns aspectos sejam discutidos, nenhuma solução específica abrangendo segurança e privacidade é proposta. Simmhan, Plale e Gannon [44] também mencionam questões de segurança de proveniência, observando que uma das aplicações de proveniência é atribuição, onde se pode verificar se dados são derivados de outros dados protegidos por direitos autorais. Os proprietários de direitos autorais, por outro lado, podem verificar quem está usando os dados que eles criaram.

No contexto da iniciativa para gerência de metadados *Dublin Core*¹, o uso de assinaturas digitais para proteger informações de autoria de metadados foi proposta por Tonkin e Allinson [144]. Como o Dublin Core utiliza XML para armazenar metadados, foi sugerido o uso do padrão *XML-Signature* [145]. No entanto, nesta tese, é mostrado que o uso de assinaturas digitais não é suficiente para proteger contra reivindicações ilegítimas de propriedade intelectual.

Em [146], Tan et al. fazem algumas considerações sobre a representação e segurança dos dados neste contexto. Observou-se que o controle de acesso é, claramente, um requisito de segurança para a proteção de dados de proveniência, algumas soluções possíveis para atender a este requisito são mencionados, tais como a classificação da sensibilidade de cada registro de proveniência, e o agrupamento dos

¹<http://dublincore.org>

usuários de acordo com os seus respectivos direitos de acesso. Observou-se também que as assinaturas digitais podem ser utilizadas para associar entidades às etapas do processo de documentação e para proteger a sua integridade, impedindo modificações acidentais ou intencionais não autorizadas.

Em [147], Braun e Shinnar realizaram estudos com usuários para levantar os requisitos de segurança para o PASS [148], um sistema de armazenamento com suporte nativo a proveniência. Eles apresentam um modelo de segurança para proveniência que usa técnicas de controle de acesso para restringir que partes de um grafo de proveniência um usuário pode ver, protegendo tanto os atributos de proveniência quanto as informações sobre ancestrais e descendentes nos processos de derivação de dados. Braun et al. [149] analisam o problema de fornecer técnicas de controle de acesso a dados de proveniência adequadas. Eles argumentam que a maioria das técnicas de controle de acesso existentes de outras áreas, tais como sistemas operacionais e bancos de dados, se concentram na proteção de conjuntos de dados, no entanto, os dados de proveniência desses objetos podem descrever relações causais, que não são adequadamente protegidas por estas técnicas de controle de acesso.

Em [150], Hasan et al. propõem uma solução de segurança com o objetivo de proteger a confidencialidade e integridade dos dados de proveniência. A solução baseia-se na criptografia de dados sensíveis de proveniência e um mecanismo de assinatura encadeada para proteger a integridade de uma cadeia de proveniência como um todo. Isso inclui bloquear ataques que busquem remover ou modificar as entradas em uma cadeia de proveniência, garantindo o não-repúdio de registros da cadeia por usuários. Os métodos utilizam criptografia assimétrica para prover confidencialidade e funções de *hash* para prover integridade.

Em [151], Dai et al. apresentam um método para avaliação de confiabilidade baseada em proveniência. Ele usa um sistema que calcula uma pontuação de confiança com base em vários parâmetros que afetam a confiabilidade. O método é útil em tomadas de decisão com base em dados, onde é importante avaliar sua qualidade e nível de confiança. O modelo de segurança baseia-se na estimativa do grau de confiabilidade tanto dos dados quanto dos seus criadores. Exemplos de critérios utilizados são os similaridade e conflito de dados, se diferentes conjuntos de dados coincidem em alguma propriedade sua pontuação de confiança mútua aumenta, uma vez que a probabilidade de estejam corretos é maior.

Em [152], Nagappan e Vouk apresentam um modelo de compartilhamento de dados de proveniência onde o usuário pode selecionar dinamicamente o nível de confidencialidade. Ele também tem o objetivo de proteger a relação de propriedade dos dados, prevenir a modificação não autorizada de dados proveniência, permitindo anotação, compartilhamento e auditoria de dados. A abordagem utiliza controle de acesso baseado em papéis para proteger a relação de propriedade dos dados. Se

Trabalho	Propriedades	Métodos
Tan et al. [146]	autenticidade, integridade	assinaturas digitais
Braun et al. [147, 149]	confidencialidade	controle de acesso
Hasan et al. [150, 154]	confidencialidade, integridade	criptografia, assinaturas digitais
Dai et al. [151]	confiabilidade	cálculo de pontuação de confiança
Nagappan et al. [152]	confidencialidade	controle de acesso
Gadelha e Mattoso [73]	autenticação, integridade	datação criptográfica, assinaturas digitais

Tabela 3.1: Comparação de abordagens para segurança de proveniência.

um usuário achar uma consulta de dados de proveniência interessante, é possível armazená-la, a fim de compartilhá-la com outros usuários.

A GSI [128, 129] é a solução de segurança subjacente de muitas das infraestruturas para e-Ciência existentes. Portanto, os sistemas de proveniência podem utilizar tais ferramentas de segurança de forma relativamente fácil para obter integridade, confidencialidade e autenticidade nas comunicações cliente-servidor via rede. Na GSI, os usuários são identificados globalmente por suas credenciais X509v3 [153], que também são utilizadas para autenticá-los. No entanto, a GSI é uma solução genérica para a segurança de redes. Para permitir a proteção da propriedade intelectual eventualmente contida nas informações de proveniência de workflows científicos são necessários mecanismos mais sofisticados, tais como assinaturas digitais e datação criptográfica na informação de proveniência que se deseja proteger. Kairos [73], apresentada no apêndice C.2, é constituída por uma arquitetura que utiliza estes mecanismos de segurança, além das já fornecidas pela GSI, para fornecer uma solução para este problema. A solução proposta é baseada em gerência de proveniência. A utilização dos mecanismos de segurança implementados na GSI, além dos que são propostos na Kairos, tem a vantagem de manter a compatibilidade com a maioria dos sistemas distribuídos existentes.

A conexão entre propriedade intelectual e sistemas de proveniência é frequentemente mencionada na literatura acadêmica, [18, 44, 71], o que sugere que informações de proveniência, que descrevem precisamente a história de experimentos científicos, seja um ativo de informação valioso que deve ser protegido. Existem diferentes abordagens existentes para se conseguir essa proteção, o mais comum nos trabalhos revisados neste seção [147, 149, 152] é utilizar mecanismos de controle de acesso para impedir o acesso não autorizado a essas informações. Isto pode ser visto como uma abordagem que visa a proteção da confidencialidade e privacidade. Hasan et al. [150] também tem como objetivo a confidencialidade dos registros de proveniência, no entanto eles usam criptografia assimétrica para obtê-la. A integridade também é ob-

tida com o utilização de assinaturas digitais. A restrição fundamental de abordagens onde somente a confidencialidade é o objetivo é a limitação da colaboração científica, pois o acesso às informações de proveniência é controlado. Esta tese propõe uma abordagem diferente ao gerar asserções verificáveis sobre quem realizou um experimento e quando isso ocorreu [73]. A asserção sobre quem realizou um experimento é feita através de assinaturas digitais e a asserção sobre quando ele ocorreu é feita através de datação criptográfica. A combinação destas duas asserções, ao contrário da abordagem proposta em [146] baseada apenas em assinaturas digitais, é essencial para provar a propriedade intelectual. Kairos [73] visa autenticar a identidade de quem gerou um registro de proveniência e quando isso ocorreu. Se houver qualquer reivindicação de propriedade intelectual conflitante, pode-se exibir um registro proveniência assinado digitalmente e datado criptograficamente para argumentar de forma contrária. Desta forma, cientistas podem compartilhar as informações de proveniência com mais antecedência, viabilizando cooperações científicas mais cedo, e com um risco bem menor em relação aos direitos de propriedade intelectual. Como a integridade dos dados é uma propriedade inerente fornecida por assinaturas digitais, Kairos também oferece esse recurso de segurança para proteção dos registros de proveniência.

Capítulo 4

Gerência de Proveniência em Workflows Científicos Paralelos e Distribuídos

Neste capítulo, apresentamos a metodologia utilizada na tese, e as contribuições realizadas ao problema de pesquisa. A descrição é feita de forma geral, os detalhes de desenvolvimento e de avaliação estão contidos nos apêndices desta tese, para os quais o leitor é direcionado nas seções seguintes.

4.1 Modelagem de Proveniência

Apresentamos no apêndice A um modelo de dados para proveniência de workflows científicos paralelos e distribuídos, que é uma extensão do OPM. Um sistema de gerência de proveniência é implementado, seguindo esse modelo, no Swift [88, 95, 96], um SGWC para ambientes paralelos e distribuídos que sucedeu o VDS (*Virtual Data System*) [42, 59, 135]. Como visto na seção 2.4, ele permite a especificação, gerência e execução de workflows científicos de larga escala, em ambientes paralelos e distribuídos. A sua linguagem de especificação de workflows possui características tais como tipos de dados, mapeadores de dados, iteradores sobre conjuntos de dados, controles condicionais de fluxo, e composição procedural. Ele permite a manipulação de conjuntos de dados com base na organização lógica dos mesmos. A notação XDTM (*XML Dataset Typing and Mapping*) [155] é usada para definir mapeadores entre esta organização lógica e a estrutura física real do conjunto de dados. As funções da linguagem de especificação realizam operações lógicas nos conjuntos dados de entrada, sem modificá-los. Ao analisar as entradas e saídas desses procedimentos, o sistema determina as dependências de dados existentes entre eles. Esta informação é usada para executar procedimentos que não têm dependências mútuas de dados em paralelo. O Swift registra uma variedade de informações sobre cada execução de workflow em seus arquivos de *log*. O sistema de gerência de proveniência exporta as informações relativas a proveniência para um banco de dados relacional por meio de ferramentas que seguem o modelo de dados apresentado. Nessa etapa, o modelo

se concentrou na coleta de informações sobre a relação entre conjuntos de dados e funções ao nível de um script Swift. Posteriormente, esse modelo foi estendido para contemplar informações sobre os processos de nível mais baixo envolvidos na execução de um workflow paralelo e distribuído com Swift, como a transferência de dados para recursos computacionais, e o envio de tarefas para os escalonadores de execução nos *clusters*.

O objetivo do apêndice A é apresentar o modelo de dados e as funcionalidades de captura de proveniência local e remota, comparando-as com o OPM. Uma avaliação comparativa é realizada com base nas funcionalidades de coleta e armazenamento de proveniência, assim como é avaliada a sua interoperabilidade com outros sistemas através do uso da OPM. As avaliações foram realizadas no âmbito do *Third Provenance Challenge* (PC3), que consistiu em implementar e executar um workflow científico LoadWorkflow do projeto Pan-STARRS [156] no Swift, coletando e armazenando informações de proveniência de sua execução, realizando as consultas de proveniência propostas no evento, e trocando de informações de proveniência com outros sistemas.

4.2 Gerência Escalável de Proveniência em Workflows Científicos Paralelos e Distribuídos

A análise do resultado da execução de um workflow científico envolve, por exemplo, examinar as entradas e saídas de cada aplicação componente do workflow, verificando se tarefas computacionais falharam em recursos computacionais remotos, e determinando todos os processos que contribuíram para a criação de um conjunto de dados em particular. No apêndice B, examinamos consultas utilizadas para explorar informações de proveniência sobre computações que envolvam muitas tarefas. Nós apresentamos um conjunto de padrões que podem ser identificados nestas consultas. Isto foi utilizado como base para a estender o modelo de proveniência proposto no apêndice A para permitir que consultas frequentes pudessem ser expressas de forma relativamente simples.

No apêndice B, abordamos o problema de gerenciar de maneira escalável a informação de proveniência de workflows compostos por um grande número de tarefas e executados em ambientes paralelos e distribuídos. Tais tarefas muitas vezes possuem dependências de dados ou de controle de fluxo, e trocam (consumo e produção) dados através de arquivos. As tarefas podem ser executadas em paralelo em vários nós computacionais, possivelmente diversos e distribuídos, como é característico de workflows científicos expressos em Swift. Nossas principais contribuições são:

1. Um modelo de dados para a representação de proveniência que estende o OPM para capturar eventos que são típicos de sistemas paralelos e distribuídos, como

uma chamada a uma tarefa computacional possivelmente ter muitas tentativas de execução devido a falhas ou por alta demanda por recursos computacionais;

2. Uma interface para consultas a proveniência que permite a visualização de grafos de proveniência e a consulta de informações de proveniência usando uma linguagem com uma sintaxe simplificada em relação ao SQL, que permite a concepção de forma relativamente simples de consultas ao abstrair em funções e procedimentos pré-construídos padrões de consulta a proveniência comumente encontrados. A interface oculta o esquema de dados subjacente através de visões abstratas do banco de dados e do cálculo automático de junções relacionais.
3. Um estudo de caso demonstrando a usabilidade desta interface de consultas e aplicações das informações de proveniência coletadas.

Este modelo de dados de proveniência e a interface de consulta, juntamente com os componentes para extração de informações de proveniência sobre execuções de scripts Swift e armazenamento das mesmas em um banco de dados relacional, foram implementados como um arcabouço de consulta a proveniência, que chamamos de MTCProv, para o sistema de scripting paralelo Swift.

Para exemplificar as análises viabilizadas pelo MTCProv implementamos um workflow científico no Swift para paralelização do método de alinhamento de seqüências BLAST [123] através do particionamento do banco de dados de entrada. Esse workflow é ilustrado na figura 4.1 e foi executado em um *cluster* de 72 nós, onde cada nó possui 8 processadores. As aplicações componentes *fastasplit* e *blastmerge* são provenientes da implementação de Mathog [157].

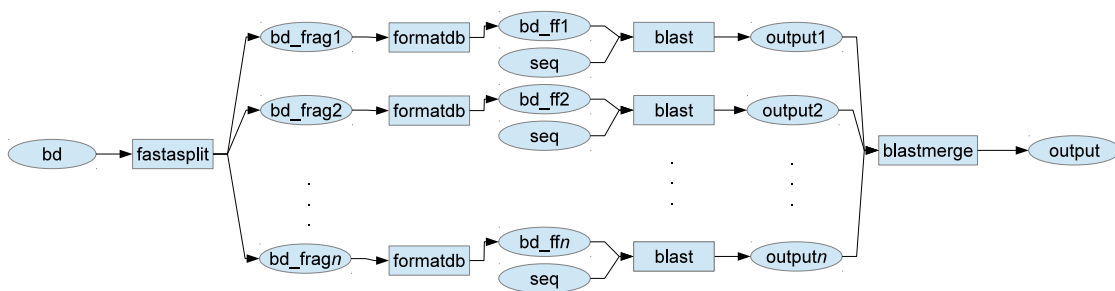


Figura 4.1: Implementação inicial do workflow de paralelização do BLAST.

Uma consulta que pode ser realizada é a verificação de utilização de CPU por uma das aplicações componentes *blast*:

```
SELECT app_exec_id, timestamp, (100 - cpu_usage) as wait
FROM runtime_info;
```

app_exec_id	timestamp	wait
blastall-3i191nsk	1339467548	100.0
blastall-3i191nsk	1339467550	81.7
blastall-3i191nsk	1339467552	50.8
blastall-3i191nsk	1339467554	45.9
blastall-3i191nsk	1339467555	38.4
blastall-3i191nsk	1339467556	31.8
blastall-3i191nsk	1339467558	30.0
blastall-3i191nsk	1339467560	34.7
blastall-3i191nsk	1339467561	33.1

Onde `wait` é a proporção do tempo de processamento não utilizado pela aplicação componente *blast*. É possível observar que a execução inicialmente consegue utilizar muito pouco do tempo de processamento, o que pode ser causado por diversos fatores, como a existência de outras aplicações em execução no mesmo processador ou a espera por operações de escrita e leitura de arquivos. As aplicações componentes *fastasplit* e *formatdb* realizam suas operações de escrita e leitura de arquivos em um sistema de armazenamento compartilhado acessível pela rede. Como estas aplicações manipulam simultaneamente arquivos relativamente grandes, o sistema de armazenamento compartilhado ficou sobrecarregado, o que foi verificado após a análise do comportamento das aplicações no próprio recurso computacional. O banco de dados de entrada do BLAST frequentemente é obtido de repositórios públicos e, nestes casos, são atualizados com uma frequência relativamente baixa. Logo, as atividades iniciais do workflow, que executam muitas operações de leitura e escrita de arquivos, são necessárias apenas quando tais bancos de dados são atualizados. Além disso, os nós do cluster possuem armazenamento local que poderia armazenar cópias destes bancos de dados utilizados frequentemente. Um refinamento do workflow, apresentado na figura 4.2, assume uma pré-distribuição dos fragmentos dos bancos de dados de entrada para eliminar as etapas iniciais do workflow.

Neste caso, como a localidade dos dados é melhorada, o impacto da gerência de dados no tempo de execução do workflow é reduzido consideravelmente. O workflow, como um todo, chega a ser executado na metade do tempo original. Executando a mesma consulta com o workflow refinado obtém-se o seguinte resultado:

```
SELECT app_exec_id, timestamp, (100 - cpu_usage) as wait
FROM runtime_info;
```

app_exec_id	timestamp	wait
blastall-leenrisk	1339924431	1.8
blastall-leenrisk	1339924437	2.2
blastall-leenrisk	1339924444	1.6

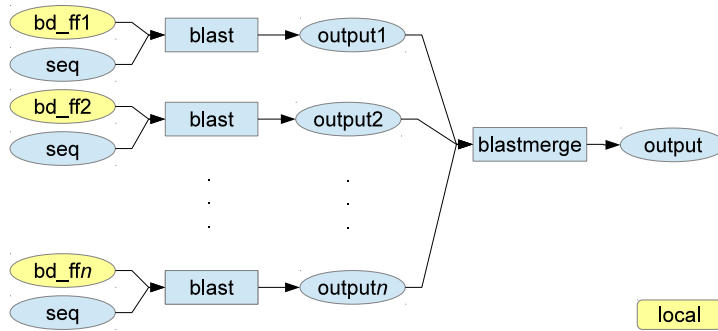


Figura 4.2: Refinamento do workflow de paralelização do BLAST.

blastall-leenrisk	1339924453	1.2
blastall-leenrisk	1339924462	0.9
blastall-leenrisk	1339924472	0.7
blastall-leenrisk	1339924483	0.5
blastall-leenrisk	1339924495	0.4
blastall-leenrisk	1339924508	0.3

O apêndice está organizado da seguinte forma. Na seção B.1, apresentamos um levantamento de consultas a proveniência frequentemente realizadas. Na seção B.2, apresentamos o modelo de dados usado para representar proveniência em computações dadas por muitas tarefas. Na seção B.3, descrevemos a concepção e implementação escalável do MTCProv, em particular, como proveniência é coletada, armazenada e consultada. Na seção B.4, avaliamos MTCProv no contexto de uma aplicação de simulação da estrutura de proteínas [97], demonstrando a sua usabilidade e utilidade.

4.3 Aspectos de Segurança de Proveniência

Informações de proveniência permitem a descrição precisa de como um experimento foi especificado, e o que aconteceu durante a sua execução, tornando mais fácil a sua reprodução para fins de verificação. Novos resultados científicos podem ser derivados a partir da análise de um experimento, o que pode produzir propriedade intelectual de valor. Portanto, esta facilidade de reprodutibilidade oferecida pelos sistemas de proveniência também pode ser vista como uma ameaça à propriedade intelectual, se mecanismos de segurança adequados não estiverem disponíveis para proteger informações de proveniência.

Os registros de proveniência são análogos aos *cadernos de laboratório* dos experimentos de bancada pois podem registrar o plano do experimento, os seus parâmetros iniciais e a descrição dos resultados. Diversas instituições publicam têm políticas oficiais a esse respeito. O Laboratório Nacional de Los Alamos [158], dos Estados Unidos, por exemplo, faz as seguintes recomendações para proteção de propriedade

intelectual contida nos cadernos de laboratório:

- “O caderno de laboratório é um dos elementos mais importantes do processo de patenteamento.”
- “... é possível perceber a importância de se manter um caderno de laboratório para certificar, e provar em tribunal se necessário, que o seu trabalho para uma invenção foi realizado anteriormente ao de outros.”
- “... as anotações devem ser assinadas e datadas por um terceiro.”
- “... após assinatura e datação, nenhuma modificação pode ser realizada.”

Proveniência segura é uma problema de pesquisa relativamente recente [146, 150, 154, 159, 160], encontrado em diferentes áreas, como workflows científicos, bancos de dados e sistemas de armazenamento. Proveniência de dados é útil em auditorias de segurança [161], e há casos em que o conteúdo dos dados de proveniência pode levar a preocupações com a privacidade [162]. Braun et al. [149] analisam o problema de fornecer técnicas adequadas de controle de acesso aos dados de proveniência, observando que estes descrevem relações causais, que não são adequadamente protegidas pelas técnicas de controle de acesso comumente utilizadas. Hasan et al. [150, 159] propõem uma solução de segurança com o objetivo de proteger a confidencialidade e integridade dos dados de proveniência. Eles usam criptografia assimétrica para obter confidencialidade e assinaturas digitais para garantir a integridade. Nagappan et al. [152] apresentam um modelo de compartilhamento de dados de proveniência que utiliza técnicas de controle de acesso baseadas em papéis desempenhados, onde o usuário seleciona dinamicamente o nível de confidencialidade. Uma abordagem comum para proteger informações de proveniência [149, 152, 163] é utilizar mecanismos de controle de acesso para prevenir acesso não autorizado a essas informações. Nenhuma dessas abordagens permite divulgação irrestrita de informações de proveniência com manutenção da atribuição, no sentido de autoria, correta, uma exigência no contexto da e-Ciência.

Muitas comunidades científicas, como as ciências da vida, são sensíveis a questões de segurança, assim a ausência de controles de segurança adequados pode impedir uma maior adoção de sistemas de gerência de proveniência em ambientes de produção científica. Nessas comunidades, a especificação do experimento e os metadados descrevendo seus resultados podem ter algum valor de propriedade intelectual associado. Os cientistas muitas vezes evitam o compartilhamento de dados de experimentos antes de publicar seus resultados em algum periódico acadêmico ou evento, para garantir a atribuição correta dos resultados científicos. Durante este intervalo, a colaboração científica é prejudicada. Portanto, os controles de segurança que impedem reivindicações ilegítimas de atribuição são um requisito de segurança

importante para os sistemas de proveniência. Esses controles devem permitir a verificação não apenas de quem executou um experimento, mas também quando ele foi executado.

O objetivo principal do apêndice C é abordar o problema de fornecer controles de segurança adequados para sistemas de proveniência para workflows científicos paralelos e distribuídos, tendo em conta os requisitos que são específicos da e-Ciência. Esses requisitos incluem, por exemplo, ser capaz de compartilhar especificações de workflows com outros cientistas, sem perder o controle sobre a propriedade intelectual. A maior exposição dos dados de proveniência em sistemas paralelos e distribuídos requer cuidados adicionais. A falta de controles de segurança adequados leva a vulnerabilidades que podem fazer com que informações de proveniência sejam acessadas sem autorização ou sejam modificadas de forma intencional ou acidental. Uma solução para este problema requer uma análise detalhada das ameaças existentes para informações de proveniência sobre as infraestruturas de computação científica distribuída atuais. Existem algumas soluções propostas para proteger as diferentes propriedades de segurança de informações de proveniência. A mais frequente é o uso de técnicas de controle de acesso para restringir quem pode ler ou modificar dados de proveniência, o que é um recurso de segurança importante, mas que não satisfaz a exigência de compartilhamento livre de informações de proveniência com garantias de proteção de propriedade intelectual. Isto se deve provavelmente ao fato de que a maioria dessas soluções foram projetadas em contextos que são diferentes da e-Ciência, que têm outros requisitos de segurança.

Em nosso esforço de modelagem de ameaças, enumeramos ameaças aos componentes de um sistema de proveniência. Muitas delas já são levadas em consideração por controles de segurança para as tecnologias subjacentes utilizadas pelos sistemas de proveniência, como bancos de dados (controle de acesso) e redes de computadores (comunicação criptografada). No apêndice C, propomos o Kairos, uma arquitetura de segurança com o objetivo de integrar protocolos de assinaturas digitais e de datação criptográfica a sistemas de proveniência para workflows científicos paralelos e distribuídos, a fim de proteger os registros científicos de proveniência. Estas técnicas são descritas na seção 2.5 e oferecem uma solução simples para garantir uma segurança robusta da atribuição de experimentos científicos computacionais.

Capítulo 5

Discussão e Conclusões

Nesta tese, apresentamos o MTCProv, um sistema de gerência de proveniência para computações científicas compostas por muitas tarefas que captura detalhes de tempo de execução de tarefas computacionais de workflows científicos paralelos e distribuídos, além da proveniência retrospectiva, relativa à derivação de dados, e prospectiva, relativa a especificações de workflow utilizadas. Descrevemos, nos parágrafos seguintes, as contribuições realizadas em relação a cada um dos objetivos traçados originalmente.

Modelagem. No apêndice A, mostramos que o modelo de proveniência proposto para workflows científicos paralelos e distribuídos, e sua respectiva implementação no sistema de gerência de proveniência do Swift, foram capazes de realizar os desafios propostos no PC3 de forma bem sucedida, demonstrando a sua capacidade de apoiar a coleta e análise de proveniência. Esse modelo é facilmente mapeado para o OPM, o que permite a interoperabilidade com outros sistemas de proveniência. Um problema que ficou evidente durante o PC3 foi o de consultas a informações de proveniência. Embora seja possível expressar as consultas de proveniência com linguagens genéricas, como SQL, nem sempre isso é fácil, devido ao uso intensivo de junções relacionais, por exemplo, e às restrições ao cálculo do fecho transitivo, onde o grafo definido pelas relações de derivação de dados é percorrido para se determinar o que precedeu uma chamada a aplicação ou um conjunto de dados. O MTCProv permite que essas informações de proveniência sejam enriquecidas com anotações específicas do domínio científico. Esse conteúdo enriquecido de proveniência, como mostrado neste trabalho, permite que consultas mais poderosas para a análise em conjunto de diferentes aspectos do experimento sejam realizadas.

Utilidade. Estudos de caso reais, com workflows científicos paralelos e distribuídos em larga escala utilizados em pesquisas correntes, demonstraram como consultas úteis se tornaram possíveis graças ao MTCProv, tal como correlações entre os parâmetros científicos e dados de consumo de recursos computacionais. No caso do workflow de paralelização do BLAST, descrito na seção 4.2, foi possível identificar através de consultas realizadas com o MTCProv que a estratégia origi-

nal de gerência dos dados era ineficiente. No caso do OOPS, apresentado na seção B.4, foi possível correlacionar o número de iterações da simulação da proteína com a qualidade da aproximação obtida, apoiando o cientista no planejamento do seu experimento levando em consideração o custo computacional envolvido.

Usabilidade. Um dos objetivos deste trabalho foi tornar o sistema de consultas a informações de proveniência, que poderia incluir uma linguagem específica de domínio [164] para consultas de proveniência, capaz de ser facilmente utilizado por cientistas para permitir o aprimoramento da sua atividade científica por meio da validação, colaboração e descoberta. O levantamento sobre os tipos de consulta definidos nas 3 edições dos *Provenance Challenges* e consultas que os cientistas gostariam de fazer sobre proveniência permitiu a definição de categorias e padrões de consultas mais frequentes. A identificação desses padrões foi importante para apoiar o projeto e implementação do MTCProv no que diz respeito à escolha de modelos adequados de dados, estratégias de armazenamento e interfaces de consulta. Problemas típicos do SQL, como a tipagem estática, foram amenizados com a implementação da interface de consultas SPQL, abstraindo os padrões de consulta frequentes com funções e procedimentos pré-construídos e automatizando junções relacionais, o que permitiu simplificar diversas consultas de proveniência.

Segurança. Este trabalho apresentou algumas técnicas para fornecer segurança aos dados de proveniência. Um dos objetivos deste trabalho foi trazer contribuições em relação ao problema proveniência segura apresentada em [154], que na sua forma geral ainda está em aberto, exigindo novas técnicas para questões controle de acesso de alta granularidade aos registros de proveniência. A proteção adequada dos dados de autoria e das informações cronológicas contidas nos registros de proveniência têm muitas aplicações, tais como a proteção dos direitos de propriedade intelectual e da análise cronológica confiável de experimentos científicos. No apêndice C, foi apresentado Kairos, uma arquitetura para permitir a verificação segura de autoria e das informações temporais em registros de proveniência. Ele utiliza técnicas de infraestruturas de chaves públicas, tais como assinaturas digitais e datação criptográfica, que são relativamente simples de implementar em SGWCs que tenham suporte a gerência de proveniência e interoperabilidade com grades. As técnicas propostas foram testadas com sucesso no Swift, demonstrando praticamente uma nova funcionalidade para proteção de informações de proveniência.

Um aspecto importante da Swift é o seu apoio à execução escalável de computações científicas em ambientes paralelos e distribuídos, juntamente com a coleta de informações de proveniência. Vários exemplos de aplicações implementadas em Swift são mencionados em [96], que incluem previsão de estrutura de proteínas, identificação de alvos de fármacos usando ancoragem computacional, e economia computacional. Em um exemplo recente, o Swift executou um workflow de análise

de imagens neurais que envolveu 500.000 tarefas computacionais. Como observado neste trabalho, o impacto da coleta e armazenamento da proveniência é pequeno em termos de tempo total de execução de *script*, tornando-o adequado para o acompanhamento de workflows científicos paralelos e distribuídos com alto grau de granularidade, mantendo a escalabilidade de execução. Os requisitos de espaço de armazenamento para de uma execução de workflow são uma proporção constante do tamanho total do arquivo de log. Resultados de extensa avaliação experimental utilizando cerca de 1000 núcleos de processamento mostrou escalabilidade, aceleração linear de desempenho ao mesmo tempo em que possibilita consultas de alto nível.

5.1 Perspectivas Futuras

Uma vez que sistemas de gestão de proveniência tornem-se robustos, úteis e usáveis, é provável que em infra-estruturas de computação distribuída de larga escala, como o TeraGrid, sejam instalados repositórios de proveniência para apoiar seus usuários. O que provavelmente levará a questões sobre a integração de dados e escalabilidade em sistemas de gerência de proveniência, que são alguns dos problemas que pretendemos investigar em etapas posteriores.

Um dos trabalhos futuros é a continuação do desenvolvimento do Swift com o objetivo de melhorar as suas funcionalidades de proveniência. Uma das questões é estudar a escalabilidade do sistema como repositório de informações de proveniência. Para casos de pequeno porte e poucos usuários, tais como o LoadWorkflow apresentado no apêndice A, isto não é um problema. Mas para grandes infraestruturas de computação, como o XSEDE [138] e o SINAPAD [165], a agregação de informações de proveniência de múltiplos experimentos executados em diversos sítios de computação poderá gerar grandes massas de dados para análise. Além disso, para computações em larga escala (que é precisamente o tipo de computações para o qual Swift é particularmente adequado) o espaço de armazenamento ocupado pelas informações de proveniência pode tornar-se muito grande, e as consultas de proveniência podem levar muito tempo para retornar resultados. É interessante proporcionar opções que possam permitir que o usuário configure o grau de detalhe das informações de proveniência, possibilitando o armazenamento de um volume menor de dados, embora (presumivelmente) com uma precisão reduzida. Também pode ser interessante avaliar a utilização técnicas de gerência de dados distribuídos, como o Mapreduce [166] e o Dremel [167], para permitir uma melhor escalabilidade.

Nossa implementação atual reúne proveniência sobre as aplicações componentes e conjuntos de dados manipulados pelo Swift durante a execução de workflows. Em relação ao problema de consultas de proveniência, pretendemos continuar a reforçar o modelo de consulta SPQL, adicionando mais funções pré-construídos, bem

como continuar a aperfeiçoar a capacidade de integrar SQL padrão em consultas SPQL. Nós planejamos permitir a geração *online* de proveniência, que permitiria a análise de proveniência durante a execução de um script Swift. Isso pode permitir a execução adaptativa de scripts, alterando dinamicamente o mapeamento de tarefas a recursos computacionais. Nossa implementação atual reúne proveniência ao nível do workflow, sobre chamadas de funções e variáveis definidas em um script Swift. Isso pode ser aumentado através da coleta de informação adicional em um nível mais baixo sobre transferências de dados e estratégias de escalonamento de tarefas. Esses registros de baixo nível podem ser associados aos respectivos registros do nível do workflow através de um esquema de abstração em camadas análogo ao apresentado em [84].

Referências Bibliográficas

- [1] KOTWANI, K., MYERS, J., BAKER, B., ET AL. “The dark energy survey data management system as a data intensive science gateway”. In: *Proceedings of the 8th International Workshop on Middleware for Grids, Clouds and e-Science*, MGC '10, p. 9:1–9:6, New York, NY, USA, 2010. ACM. ISBN: 978-1-4503-0453-5. doi: 10.1145/1890799.1890808. Disponível em: <<http://doi.acm.org/10.1145/1890799.1890808>>.
- [2] OINN, T., LI, P., KELL, D., ET AL. “Taverna/myGrid: Aligning a Workflow System with the Life Sciences Community”. In: Taylor, I., Deelman, E., Gannon, D., et al. (Eds.), *Workflows for e-Science*, Springer, pp. 300–319, 2007. ISBN: 978-1-84628-757-2.
- [3] FREIRE, J., SILVA, C., CALLAHAN, S., ET AL. “Managing Rapidly-Evolving Scientific Workflows”. In: *Provenance and Annotation of Data*, v. 4145, *Lecture Notes in Computer Science*, Springer, pp. 10–18, 2006. ISBN: 978-3-540-46302-3, 978-3-540-46303-0. Disponível em: <<http://www.springerlink.com/content/u86053504x624585/>>.
- [4] MOREAU, L., FREIRE, J., FUTRELLE, J., ET AL. *The Open Provenance Model*. Relatório técnico, University of Southampton, 2007.
- [5] “Third Provenance Challenge”. 2009. Disponível em: <<http://twiki.ipaw.info/bin/view/Challenge/ThirdProvenanceChallenge>>.
- [6] “Genetic Sequence Data Bank, Release 189.0, Distribution Release Notes”. abr. 2012. Disponível em: <<ftp://ftp.ncbi.nih.gov/genbank/gbrel.txt>>.
- [7] COHEN, J. “Bioinformatics: an introduction for computer scientists”, *ACM Comput. Surv.*, v. 36, n. 2, pp. 122–158, jun. 2004. ISSN: 0360-0300. doi: 10.1145/1031120.1031122. Disponível em: <<http://doi.acm.org/10.1145/1031120.1031122>>.

- [8] “CERN experiments observe particle consistent with long-sought Higgs boson”. jul. 2012. Disponível em: <http://press.web.cern.ch/press/PressReleases/Releases2012/PR17.12E.html>.
- [9] DEWDNEY, P., HALL, P., SCHILIZZI, R., ET AL. “The Square Kilometre Array”, *Proceedings of the IEEE*, v. 97, n. 8, pp. 1482–1496, ago. 2009. ISSN: 0018-9219. doi: 10.1109/JPROC.2009.2021005.
- [10] “Astronomical Data Deluge”, *The Netherlands Institute for Radio Astronomy (ASTRON) and IBM Center for Exascale Technology*, 2012.
- [11] KOUZES, R. T., ANDERSON, G. A., ELBERT, S. T., ET AL. “The Changing Paradigm of Data-Intensive Computing”, *Computer*, v. 42, n. 1, pp. 26–34, jan. 2009. ISSN: 0018-9162. doi: 10.1109/MC.2009.26.
- [12] SZALAY, A. S., BELL, G., VANDENBERG, J., ET AL. “GrayWulf: Scalable Clustered Architecture for Data Intensive Computing”. In: *42nd Hawaii International Conference on System Sciences, 2009. HICSS '09*, pp. 1–10. IEEE, jan. 2009. ISBN: 978-0-7695-3450-3. doi: 10.1109/HICSS.2009.234.
- [13] FOSTER, I., KESSELMAN, C., TUECKE, S. “The Anatomy of the Grid: Enabling Scalable Virtual Organizations”, *International Journal of High Performance Computing Applications*, v. 15, n. 3, pp. 200–222, 2001. doi: 10.1177/109434200101500302. Disponível em: <http://hpc.sagepub.com/content/15/3/200.abstract>.
- [14] FOSTER, I. “Service-Oriented Science”, *Science*, v. 308, n. 5723, pp. 814–817, maio 2005. doi: 10.1126/science.1110411. Disponível em: <http://www.sciencemag.org/content/308/5723/814.abstract>.
- [15] OLIVEIRA, D., BAIÃO, F. A., MATTOSO, M. “Towards a Taxonomy for Cloud Computing from an e-Science Perspective”. In: Antonopoulos, N., Gillam, L. (Eds.), *Cloud Computing*, Computer Communications and Networks, Springer London, pp. 47–62, 2010. ISBN: 978-1-84996-241-4.
- [16] ALLEN, B., BRESNAHAN, J., CHILDERS, L., ET AL. “Software as a service for data scientists”, *Commun. ACM*, v. 55, n. 2, pp. 81–88, fev. 2012. ISSN: 0001-0782. doi: 10.1145/2076450.2076468. Disponível em: <http://doi.acm.org/10.1145/2076450.2076468>.
- [17] CHERVENAK, A., FOSTER, I., KESSELMAN, C., ET AL. “The Data Grid: Towards an Architecture for the Distributed Management and Analysis

of Large Scientific Datasets”, *Journal of Network and Computer Applications*, v. 23, pp. 200, 2001.

- [18] PACITTI, E., VALDURIEZ, P., MATTOSO, M. “Grid Data Management: Open Problems and New Issues”, *Journal of Grid Computing*, v. 5, n. 3, pp. 273–281, 2007. ISSN: 1570-7873, 1572-9184. doi: 10.1007/s10723-007-9081-9. Disponível em: <<http://www.springerlink.com/content/6q67772276347825/>>.
- [19] DEELMAN, E., CHERVENAK, A. “Data Management Challenges of Data-Intensive Scientific Workflows”. In: *Proc. Eighth IEEE International Symposium on Cluster Computing and the Grid*, p. 692, 2008.
- [20] AILAMAKI, A., KANTERE, V., DASH, D. “Managing scientific data”, *Commun. ACM*, v. 53, n. 6, pp. 68–78, jun. 2010. ISSN: 0001-0782. doi: 10.1145/1743546.1743568. Disponível em: <<http://doi.acm.org/10.1145/1743546.1743568>>.
- [21] BARU, C., MOORE, R., RAJASEKAR, A., ET AL. “The SDSC Storage Resource Broker”. In: *Proc. 1998 Conference of the IBM Centre for Advanced Studies on Collaborative Research*, 1998.
- [22] HEDGES, M., HASAN, A., BLANKE, T. “Management and Preservation of Research Data with iRODS”. In: *Proc. 1st ACM Workshop on CyberInfrastructure: Information Management in eScience*, p. 22, 2007.
- [23] ALLCOCK, B., BESTER, J., BRESNAHAN, J., ET AL. “Data management and transfer in high-performance computational grid environments”, *Parallel Computing*, v. 28, n. 5, pp. 771, 2002.
- [24] CHERVENAK, A., PALAVALLI, N., BHARATHI, S., ET AL. “Performance and Scalability of a Replica Location Service”. In: *Proc. 13th IEEE International Symposium on High Performance Distributed Computing (HPDC 2004)*, p. 191, 2004.
- [25] FOSTER, I., KESSELMAN, C. “Globus: a Metacomputing Infrastructure Toolkit”, *International Journal of High Performance Computing Applications*, v. 11, n. 2, pp. 115–128, jun. 1997. ISSN: 1094-3420, 1741-2846. doi: 10.1177/109434209701100205. Disponível em: <<http://hpc.sagepub.com/content/11/2/115>>.
- [26] LIU, W., TIEMAN, B., KETTIMUTHU, R., ET AL. “Moving huge scientific datasets over the Internet”, *Concurrency and Computation: Practice and*

Experience, v. 23, n. 18, pp. 2404–2420, dez. 2011. ISSN: 1532-0634. doi: 10.1002/cpe.1779.

- [27] FOSTER, I., VÖCKLER, J., WILDE, M., ET AL. “The Virtual Data Grid: A New Model and Architecture for Data-Intensive Collaboration”. In: *Proceedings of the 2003 CIDR Conference*, 2003. Disponível em: <<http://www.cidrdb.org/cidr2003/program/p18.pdf>>.
- [28] GRAY, J., HEY, T., TANSLEY, S., ET AL. “Jim Gray on eScience: A Transformed Scientific Method”. In: *The Fourth Paradigm: Data-Intensive Scientific Discovery*, Microsoft Research, pp. xvii–xxxii, 2009.
- [29] BELL, G., HEY, T., SZALAY, A. “Beyond the Data Deluge”, *Science*, v. 323, n. 5919, pp. 1297–1298, mar. 2009. doi: 10.1126/science.1170411. Disponível em: <<http://www.sciencemag.org/content/323/5919/1297.short>>.
- [30] BRYANT, R. E. “Data-Intensive Scalable Computing for Scientific Applications”, *Computing in Science & Engineering*, v. 13, n. 6, pp. 25–33, dez. 2011. ISSN: 1521-9615. doi: 10.1109/MCSE.2011.73.
- [31] SZALAY, A. “Extreme Data-Intensive Scientific Computing”, *Computing in Science & Engineering*, v. 13, n. 6, pp. 34–41, dez. 2011. ISSN: 1521-9615. doi: 10.1109/MCSE.2011.74.
- [32] NEWMAN, H., ELLISMAN, M., ORCUTT, J. “Data-Intensive E-Science Frontier Research”, *Communications of the ACM*, v. 46, n. 11, pp. 77, 2003.
- [33] HEY, T., TREFETHEN, A. “Cyberinfrastructure for e-Science”, *Science*, v. 308, pp. 821, 2005.
- [34] DAVIDSON, S. B., FREIRE, J. “Provenance and scientific workflows: challenges and opportunities”. In: *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, SIGMOD '08, p. 1345–1350, New York, NY, USA, 2008. ACM. ISBN: 978-1-60558-102-6. doi: 10.1145/1376616.1376772. Disponível em: <<http://doi.acm.org/10.1145/1376616.1376772>>.
- [35] Taylor, I., Deelman, E., Gannon, D., et al. (Eds.). *Workflows for e-Science: Scientific Workflows for Grids*. 2007.
- [36] GIL, Y., DEELMAN, E., ELLISMAN, M., ET AL. “Examining the Challenges of Scientific Workflows”, *IEEE Computer*, v. 40, n. 12, pp. 32, 2007.

- [37] ZHAO, Y., RAICU, I., FOSTER, I. “Scientific Workflow Systems for 21st Century, New Bottle or New Wine?” In: *Proc. IEEE Congress on Services - Part I*, pp. 471, IEEE Computer Society, 2008. ISBN: 978-0-7695-3286-8.
- [38] DEELMAN, E., GANNON, D., SHIELDS, M., ET AL. “Workflows and e-Science: An overview of workflow system features and capabilities”, *Future Generation Computer Systems*, v. 25, n. 5, pp. 528–540, 2009. ISSN: 0167-739X. doi: 10.1016/j.future.2008.06.012.
- [39] LUDÄSCHER, B., ALTINTAS, I., BOWERS, S., ET AL. “Scientific Process Automation and Workflow Management”. In: *Scientific Data Management Challenges, Technology, and Deployment*, Chapman & Hall, pp. 467–508, 2010.
- [40] VAN DER AALST, W., VAN HEE, K. *Workflow Management: Models, Methods, and Systems*. Cambridge, MA, USA, MIT Press, 2004. ISBN: 0262720469.
- [41] BARGA, R., GANNON, D. “Scientific versus Business Workflows”. In: Taylor, I. J., Deelman, E., Gannon, D. B., et al. (Eds.), *Workflows for e-Science*, Springer, pp. 9–16, 2007. ISBN: 978-1-84628-757-2.
- [42] FOSTER, I., VÖCKLER, J., WILDE, M., ET AL. “Chimera: A Virtual Data System for Representing, Querying, and Automating Data Derivation”. In: *Proc. International Conference on Scientific and Statistical Database Management (SSDBM 2002)*, pp. 37–46. IEEE Computer Society, 2002.
- [43] BOSE, R., FREW, J. “Lineage retrieval for scientific data processing: a survey”, *ACM Comput. Surv.*, v. 37, n. 1, pp. 1–28, mar. 2005. ISSN: 0360-0300. doi: 10.1145/1057977.1057978. Disponível em: <<http://doi.acm.org/10.1145/1057977.1057978>>.
- [44] SIMMHAN, Y., PLALE, B., GANNON, D. “A survey of data provenance in e-science”, *SIGMOD Rec.*, v. 34, n. 3, pp. 31–36, set. 2005. ISSN: 0163-5808. doi: 10.1145/1084805.1084812. Disponível em: <<http://doi.acm.org/10.1145/1084805.1084812>>.
- [45] MILES, S., GROTH, P., BRANCO, M., ET AL. “The Requirements of Using Provenance in e-Science Experiments”, *Journal of Grid Computing*, v. 5, n. 1, pp. 1–25, dez. 2006. ISSN: 1570-7873, 1572-9184. doi: 10.1007/s10723-006-9055-3. Disponível em: <<http://www.springerlink.com/content/8257g3g11571n271/>>.

- [46] FREIRE, J., KOOP, D., SANTOS, E., ET AL. “Provenance for Computational Tasks: A Survey”, *Computing in Science & Engineering*, v. 10, n. 3, pp. 11–21, 2008. ISSN: 1521-9615.
- [47] MOREAU, L., GROTH, P., MILES, S., ET AL. “The Provenance of Electronic Data”, *Communications of the ACM*, v. 51, n. 4, pp. 58, 2008.
- [48] GROTH, P., MOREAU, L. “Recording Process Documentation for Provenance”, *IEEE Transactions on Parallel and Distributed Systems*, v. 20, n. 9, pp. 1246–1259, set. 2009. ISSN: 1045-9219. doi: 10.1109/TPDS.2008.215.
- [49] DA CRUZ, S., CAMPOS, M., MATTOSO, M. “Towards a Taxonomy of Provenance in Scientific Workflow Management Systems”. In: *Proc. IEEE Congress on Services, Part I, (SERVICES I 2009)*, pp. 259–266, 2009.
- [50] DEELMAN, E., BERRIMAN, B., CHERVENAK, A., ET AL. “Metadata and Provenance Management”. In: *Scientific Data Management Challenges, Technology, and Deployment*, Chapman & Hall, pp. 433–466, 2010.
- [51] MOREAU, L., CLIFFORD, B., FREIRE, J., ET AL. “The Open Provenance Model core specification (v1.1)”, *Future Generation Computer Systems*, v. 27, n. 6, pp. 743–756, 2011. ISSN: 0167-739X. doi: 10.1016/j.future.2010.07.005.
- [52] MILES, S., WONG, S. C., FANG, W., ET AL. “Provenance-based validation of e-science experiments”, *Web Semantics: Science, Services and Agents on the World Wide Web*, v. 5, n. 1, pp. 28–38, mar. 2007. ISSN: 1570-8268. doi: 10.1016/j.websem.2006.11.003.
- [53] FREIRE, J., SILVA, C. T. “Making Computations and Publications Reproducible with VisTrails”, *Computing in Science Engineering*, v. 14, n. 4, pp. 18–25, ago. 2012. ISSN: 1521-9615. doi: 10.1109/MCSE.2012.76.
- [54] FREIRE, J., BONNET, P., SHASHA, D. “Computational reproducibility: state-of-the-art, challenges, and database research opportunities”. In: *Proceedings of the 2012 international conference on Management of Data, SIGMOD ’12*, p. 593–596, New York, NY, USA, 2012. ACM. ISBN: 978-1-4503-1247-9. doi: 10.1145/2213836.2213908. Disponível em: <<http://doi.acm.org/10.1145/2213836.2213908>>.
- [55] ALTINTAS, I., BARNEY, O., JAEGER-FRANK, E. “Provenance Collection Support in the Kepler Scientific Workflow System”. In: *Proc 1st Inter-*

national Provenance and Annotation Workshop (IPAW 2006), v. 4145, 2006.

- [56] STEVENS, R., ZHAO, J., GOBLE, C. “Using provenance to manage knowledge of In Silico experiments”, *Briefings in Bioinformatics*, v. 8, n. 3, pp. 183–194, maio 2007. doi: 10.1093/bib/bbm015. Disponível em: <<http://bib.oxfordjournals.org/content/8/3/183.abstract>>.
- [57] BITON, O., COHEN-BOULAKIA, S., DAVIDSON, S. B. “Zoom*UserViews: querying relevant provenance in workflow systems”. In: *Proceedings of the 33rd international conference on Very large data bases, VLDB '07*, p. 1366–1369. VLDB Endowment, 2007. ISBN: 978-1-59593-649-3. Disponível em: <<http://dl.acm.org/citation.cfm?id=1325851.1326016>>.
- [58] BOWERS, S., MCPHILLIPS, T. M., LUDÄSCHER, B. “Provenance in collection-oriented scientific workflows”, *Concurrency and Computation: Practice and Experience*, v. 20, n. 5, pp. 519–529, abr. 2008. ISSN: 1532-0634. doi: 10.1002/cpe.1226.
- [59] CLIFFORD, B., FOSTER, I., VOECKLER, J., ET AL. “Tracking Provenance in a Virtual Data Grid”, *Concurrency and Computation: Practice and Experience*, v. 20, n. 5, pp. 575, 2008.
- [60] DA CRUZ, S., BARROS, P., BISCH, P., ET AL. “Provenance Services for Distributed Workflows”. In: *Proc. 8th IEEE International Symposium on Cluster Computing and the Grid (CCGrid 2008)*, p. 533, 2008.
- [61] HOLLAND, D. A., SELTZER, M. I., BRAUN, U., ET AL. “PASSing the provenance challenge”, *Concurrency and Computation: Practice and Experience*, v. 20, n. 5, pp. John Wiley & Sons, Ltd.–540, 2008. Disponível em: <<http://dx.doi.org/10.1002/cpe.1227>>.
- [62] ZHAO, J., GOBLE, C., STEVENS, R., ET AL. “Mining Taverna’s semantic web of provenance”, *Concurrency and Computation: Practice and Experience*, v. 20, n. 5, pp. 463–472, abr. 2008. ISSN: 1532-0634. doi: 10.1002/cpe.1231.
- [63] MARINHO, A., MURTA, L., WERNER, C., ET AL. “A Strategy for Provenance Gathering in Distributed Scientific Workflows”. In: *IEEE 3rd International Workshop on Scientific Workflows (SWF 2009). Proc. 7th IEEE International Conference on Web Services (ICWS 2009)*, p. 347, 2009.

- [64] GADELHA, L., CLIFFORD, B., MATTOSO, M., ET AL. “Provenance management in Swift”, *Future Generation Computer Systems*, v. 27, n. 6, pp. 775–780, 2011. Disponível em: <<http://dx.doi.org/10.1016/j.future.2010.05.003>>.
- [65] MARINHO, A., MURTA, L., WERNER, C., ET AL. “ProvManager: a provenance management system for scientific workflows”, *Concurrency and Computation: Practice and Experience*, v. 24, n. 13, pp. 1513–1530, 2011. ISSN: 1532-0634. doi: 10.1002/cpe.1870.
- [66] MOREAU, L. E. A. “Special Issue: The First Provenance Challenge”, *Concurrency and Computation: Practice and Experience*, v. 20, n. 5, pp. John Wiley & Sons, Ltd.–418, 2008. Disponível em: <<http://dx.doi.org/10.1002/cpe.1233>>.
- [67] SIMMHAN, Y., GROTH, P., MOREAU, L. “Special Section: The third provenance challenge on using the open provenance model for interoperability”, *Future Generation Computer Systems*, v. 27, n. 6, pp. 737–742, jun. 2011. ISSN: 0167-739X. doi: 10.1016/j.future.2010.11.020.
- [68] MALIK, T., NISTOR, L., GEHANI, A. “Tracking and Sketching Distributed Data Provenance”. In: *Proc. 6th IEEE International Conference on e-Science (e-Science 2010)*, 2010.
- [69] BENABDELKADER, A., SANTCROOS, M., MADOUGOU, S., ET AL. “A Provenance Approach to Trace Scientific Experiments on a Grid Infrastructure”. In: *2011 IEEE 7th International Conference on E-Science (e-Science)*, pp. 134–141. IEEE, dez. 2011. ISBN: 978-1-4577-2163-2. doi: 10.1109/eScience.2011.27.
- [70] GROTH, P., MOREAU, L. “Representing distributed systems using the Open Provenance Model”, *Future Generation Computer Systems*, v. 27, n. 6, pp. 757–765, jun. 2011. ISSN: 0167-739X. doi: 10.1016/j.future.2010.10.001.
- [71] MILES, S., GROTH, P., BRANCO, M., ET AL. “The Requirements of Recording and Using Provenance in e-Science”, *Journal of Grid Computing*, v. 5, n. 1, pp. 1–25, 2007.
- [72] SZOMSZOR, M., MOREAU, L. “Recording and Reasoning over Data Provenance in Web and Grid Services”. In: *Proc. International Conference on Ontologies, Databases and Applications of SEmantics (ODBASE’03)*, v. 2888, pp. 620, Springer, 2003.

- [73] GADELHA, L., MATTOSO, M. “Kairos: An Architecture for Securing Authorship and Temporal Information of Provenance Data in Grid-Enabled Workflow Management Systems”. In: *IEEE Fourth International Conference on eScience (e-Science 2008)*, pp. 597–602. IEEE, 2008. ISBN: 978-1-4244-3380-3. doi: 10.1109/eScience.2008.161. Disponível em: <<http://dx.doi.org/10.1109/eScience.2008.161>>.
- [74] DAVIDSON, S., KHANNA, S., ROY, S., ET AL. “Privacy Issues in Scientific Workflow Provenance”. In: *Proceedings of the 1st International Workshop on Workflow Approaches to New Data-centric Science (WANDS 2010)*, 2010.
- [75] ANAND, M. K., BOWERS, S., LUDASCHER, B. “Provenance browser: Displaying and querying scientific workflow provenance graphs”. In: *2010 IEEE 26th International Conference on Data Engineering (ICDE)*, pp. 1201–1204. IEEE, mar. 2010. ISBN: 978-1-4244-5445-7. doi: 10.1109/ICDE.2010.5447741.
- [76] BARGA, R. S., DIGIAMPIETRI, L. A. “Automatic capture and efficient storage of e-Science experiment provenance”, *Concurrency and Computation: Practice and Experience*, v. 20, n. 5, pp. 419–429, 2008. ISSN: 1532-0634. doi: 10.1002/cpe.1235.
- [77] GOLBECK, J., HENDLER, J. “A Semantic Web approach to the provenance challenge”, *Concurrency and Computation: Practice and Experience*, v. 20, n. 5, pp. 431–439, 2008. ISSN: 1532-0634. doi: 10.1002/cpe.1238.
- [78] SIMMHAN, Y., PLALE, B., GANNON, D. “Query capabilities of the Karma provenance framework”, *Concurrency and Computation: Practice and Experience*, v. 20, n. 5, pp. 441–451, 2008. ISSN: 1532-0634. doi: 10.1002/cpe.1229.
- [79] ZHAO, Y., LU, S. “A Logic Programming Approach to Scientific Workflow Provenance Querying”. In: *Provenance and Annotation of Data and Processes (IPAW 2008)*, v. 5272, *Lecture Notes in Computer Science*, pp. 31–44. Springer, 2008. Disponível em: <http://dx.doi.org/10.1007/978-3-540-89965-5_5>.
- [80] DEY, S., KÖHLER, S., BOWERS, S., ET AL. “Datalog as a Lingua Franca for Provenance Querying and Reasoning”. In: *Proc. 4th USENIX Workshop on Theory and Practice of Provenance (TaPP'12)*, 2012.

- [81] SCHEIDEGGER, C., KOOP, D., SANTOS, E., ET AL. “Tackling the Provenance Challenge One Layer at a Time”, *Concurrency and Computation: Practice and Experience*, v. 20, n. 5, pp. 473–483, 2008.
- [82] AMSTERDAMER, Y., DAVIDSON, S., DEUTCH, D., ET AL. “Putting Lips-tick on Pig: Enabling Database-style Workflow Provenance”, *Proceedings of the VLDB Endowment*, v. 5, n. 4, pp. 346–357, 2011.
- [83] ANAND, M., BOWERS, S., MCPHILLIPS, T., ET AL. “Exploring Scientific Workflow Provenance Using Hybrid Queries over Nested Data and Lineage Graphs”. In: *Scientific and Statistical Database Management*, v. 5566, *Lecture Notes in Computer Science*, Springer, pp. 237–254, 2009. ISBN: 978-3-642-02278-4, 978-3-642-02279-1. Disponível em: <<http://www.springerlink.com/content/k45j02238167543q/>>.
- [84] MUNISWAMY-REDDY, K.-K., BRAUN, U., HOLLAND, D. A., ET AL. “Layering in provenance systems”. In: *Proceedings of the 2009 conference on USENIX Annual technical conference*, USENIX’09, p. 10–10, Berkeley, CA, USA, 2009. USENIX Association. Disponível em: <<http://dl.acm.org/citation.cfm?id=1855807.1855817>>.
- [85] ANAND, M. K., BOWERS, S., ALTINTAS, I., ET AL. “Approaches for Exploring and Querying Scientific Workflow Provenance Graphs”. In: Hutchison, D., Kanade, T., Kittler, J., et al. (Eds.), *Provenance and Annotation of Data and Processes*, v. 6378, Springer Berlin Heidelberg, pp. 17–26, Berlin, Heidelberg, 2010. ISBN: 978-3-642-17818-4, 978-3-642-17819-1. Disponível em: <<http://www.springerlink.com/content/p58k34u063125315/>>.
- [86] LIM, C., LU, S., CHEBOTKO, A., ET AL. “OPQL: A First OPM-Level Query Language for Scientific Workflow Provenance”. In: *Services Computing, IEEE International Conference on*, pp. 136–143, Los Alamitos, CA, USA, 2011. IEEE Computer Society. ISBN: 978-0-7695-4462-5. doi: <http://doi.ieeecomputersociety.org/10.1109/SCC.2011.60>.
- [87] RAICU, I., FOSTER, I. T., ZHAO, Y. “Many-task computing for grids and supercomputers”. In: *Workshop on Many-Task Computing on Grids and Supercomputers, 2008. MTAGS 2008*, pp. 1–11. IEEE, nov. 2008. ISBN: 978-1-4244-2872-4. doi: 10.1109/MTAGS.2008.4777912.
- [88] WILDE, M., HATEGAN, M., WOZNIAK, J., ET AL. “Swift: A language for distributed parallel scripting”, *Parallel Computing*, v. 37, n. 9, pp. 634–652, 2011.

- [89] SERRA DA CRUZ, S. M., DA SILVA, F. N., GADELHA, L. M., ET AL. “A Lightweight Middleware Monitor for Distributed Scientific Workflows”. In: *8th IEEE International Symposium on Cluster Computing and the Grid, 2008. CCGRID '08*, pp. 693–698. IEEE, maio 2008. ISBN: 978-0-7695-3156-4. doi: 10.1109/CCGRID.2008.89.
- [90] GADELHA, L. M. R., MATTOSO, M., WILDE, M., ET AL. “Towards a Threat Model for Provenance in e-Science”. In: Hutchison, D., Kanade, T., Kittler, J., et al. (Eds.), *Provenance and Annotation of Data and Processes*, v. 6378, Springer Berlin Heidelberg, pp. 277–279, Berlin, Heidelberg, 2010. ISBN: 978-3-642-17819-1. Disponível em: <http://dx.doi.org/10.1007/978-3-642-17819-1_32>.
- [91] GADELHA, L., CLIFFORD, B., MATTOSO, M., ET AL. *Provenance Management in Swift with Implementation Details*. Relatório Técnico ANL/MCS-TM-311, Argonne National Laboratory, 2010.
- [92] GADELHA, L., MATTOSO, M., WILDE, M., ET AL. “Provenance Query Patterns for Many-Task Scientific Computations”. In: *Proceedings of the 3rd USENIX Workshop on Theory and Applications of Provenance (TaPP'11)*, 2011.
- [93] GADELHA, L., WILDE, M., MATTOSO, M., ET AL. “Exploring provenance in high performance scientific computing”. In: *Proceedings of the First Annual Workshop on High Performance Computing Meets Databases, HPCDB '11*, p. 17–20, New York, NY, USA, 2011. ACM. ISBN: 978-1-4503-1157-1. doi: 10.1145/2125636.2125643. Disponível em: <<http://doi.acm.org/10.1145/2125636.2125643>>.
- [94] GADELHA, L., WILDE, M., MATTOSO, M., ET AL. “MTCProv: a practical provenance query framework for many-task scientific computing”, *Distributed and Parallel Databases*, v. 30, n. 5-6, pp. 351–370, 2012. ISSN: 0926-8782. doi: 10.1007/s10619-012-7104-4.
- [95] ZHAO, Y., HATEGAN, M., CLIFFORD, B., ET AL. “Swift: Fast, Reliable, Loosely Coupled Parallel Computation”. In: *Proc. 1st IEEE International Workshop on Scientific Workflows (SWF 2007)*, pp. 199–206, 2007.
- [96] WILDE, M., FOSTER, I., ISKRA, K., ET AL. “Parallel Scripting for Applications at the Petascale and Beyond”, *Computer*, v. 42, n. 11, pp. 50–60, 2009.

- [97] ADHIKARI, A., PENG, J., WILDE, M., ET AL. “Modeling large regions in proteins: Applications to loops, termini, and folding”, *Protein Science*, v. 21, n. 1, pp. 107–121, 2012. Disponível em: <<http://dx.doi.org/10.1002/pro.767>>.
- [98] MATTOSO, M., WERNER, C., TRAVASSOS, G., ET AL. “Gerenciando Experimentos Científicos em Larga Escala”. In: *Anais do XXVIII Congresso da SBC*, p. 135, 2008.
- [99] MATTOSO, M., WERNER, C., TRAVASSOS, G., ET AL. “Desafios no apoio à composição de experimentos científicos em larga escala”. In: *XXXVI Seminário Integrado de Software e Hardware (SEMISH), XXIX Congresso da Sociedade Brasileira de Computação (CSBC)*, pp. 307–321, 2009.
- [100] MATTOSO, M., WERNER, C., TRAVASSOS, G., ET AL. “Towards supporting the life cycle of large scale scientific experiments”, *International Journal of Business Process Integration and Management*, v. 5, n. 1, pp. 79–92, jan. 2010. doi: 10.1504/IJBPIIM.2010.033176. Disponível em: <<http://dx.doi.org/10.1504/IJBPIIM.2010.033176>>.
- [101] KOOP, D., SANTOS, E., MATES, P., ET AL. “A provenance-based infrastructure to support the life cycle of executable papers”, *Procedia Computer Science*, v. 4, pp. 648–657, 2011.
- [102] GOBLE, C. A., BHAGAT, J., ALEKSEJEVS, S., ET AL. “myExperiment: a repository and social network for the sharing of bioinformatics workflows”, *Nucleic Acids Research*, v. 38, n. Web Server, pp. W677–W682, maio 2010. ISSN: 0305-1048, 1362-4962. doi: 10.1093/nar/gkq429.
- [103] AHUJA, S., CARRIERO, N., GELERNTER, D. “Linda and Friends”, *Computer*, v. 19, n. 8, pp. 26–34, ago. 1986. ISSN: 0018-9162. doi: 10.1109/MC.1986.1663305.
- [104] CARRIERO, N., GELERNTER, D. “Linda in context”, *Commun. ACM*, v. 32, n. 4, pp. 444–458, abr. 1989. ISSN: 0001-0782. doi: 10.1145/63334.63337. Disponível em: <<http://doi.acm.org/10.1145/63334.63337>>.
- [105] GELERNTER, D., CARRIERO, N. “Coordination languages and their significance”, *Commun. ACM*, v. 35, n. 2, pp. 97–107, fev. 1992. ISSN: 0001-0782. doi: 10.1145/129630.129635. Disponível em: <<http://doi.acm.org/10.1145/129630.129635>>.

- [106] FOSTER, I., TAYLOR, S. *Strand: New Concepts in Parallel Programming*. USA, Prentice-Hall, 1990.
- [107] FOSTER, I. “Compositional parallel programming languages”, *ACM Trans. Program. Lang. Syst.*, v. 18, n. 4, pp. 454–476, jul. 1996. ISSN: 0164-0925. doi: 10.1145/233561.233565. Disponível em: <<http://doi.acm.org/10.1145/233561.233565>>.
- [108] OGASAWARA, E., MURTA, L., WERNER, C., ET AL. “Experiment Line: Software Reuse in Scientific Workflows”. In: *International Conference on Scientific and Statistical Database Management (SSDBM 2009)*, v. 5566, pp. 272, Springer, 2009.
- [109] MEYER, L., ANNIS, J., WILDE, M., ET AL. “Planning Spatial Workflow to Optimize Grid Performance”. In: *ACM Symposium of Applied Computing*, p. 790, 2006.
- [110] MEYER, L., WILDE, M., MATTOSO, M., ET AL. “An Opportunistic Algorithm for Scheduling Workflows on Grids”. In: *Proc. International Conference on High Performance Computing for Computational Science (VECPAR)*, v. 4395, p. 224, 2007.
- [111] OGASAWARA, E., DE OLIVEIRA, D., VALDURIEZ, P., ET AL. “An Algebraic Approach for Data-Centric Scientific Workflows”, *Proceedings of the VLDB Endowment*, v. 4, n. 12, pp. 1339, 2011.
- [112] OINN, T., ADDIS, M., FERRIS, J., ET AL. “Taverna: a tool for the composition and enactment of bioinformatics workflows”, *Bioinformatics*, v. 20, n. 17, pp. 3045–3054, 2004.
- [113] DEELMAN, E., GURMEET, G., SU, M.-H., ET AL. “Pegasus: A framework for mapping complex scientific workflows onto distributed systems”, *Scientific Programming*, v. 13, n. 3, pp. 219–237, 2005. Disponível em: <<http://iospress.metapress.com/content/84H5Q70AWX6FAU0W>>.
- [114] LUDÄSCHER, B., ALTINTAS, I., BERKLEY, C., ET AL. “Scientific Workflow Management and the Kepler System”, *Concurrency and Computation: Practice and Experience*, v. 18, n. 10, pp. 1065, 2006.
- [115] EKER, J., JANNECK, J. W., LEE, E. A., ET AL. “Taming Heterogeneity - the Ptolemy Approach”, *Proc. IEEE*, v. 91, n. 1, pp. 144, 2003.

- [116] FAHRINGER, T., PRODAN, R., DUAN, R., ET AL. “ASKALON: A Development and Grid Computing Environment for Scientific Workflows”. In: *Workflows for e-Science*, pp. 471, Springer, 2007.
- [117] VAN DER AALST, W., TER HOFSTEDE, A., KIEPUSZEWSKI, B., ET AL. “Workflow Patterns”, *Distributed and Parallel Databases*, v. 14, n. 1, pp. 5–51, 2003. ISSN: 0926-8782. doi: 10.1023/A:1022883727209.
- [118] RUSSELL, N., ARTHUR, VAN DER AALST, W., ET AL. *Workflow Control-Flow Patterns: A Revised View*. Relatório técnico, BPM Center Report, 2006.
- [119] “Workflow Patterns”. 2012. Disponível em: <http://www.workflowpatterns.com>.
- [120] YU, J., BUYYA, R. “A Taxonomy of Scientific Workflow Systems for Grid Computing”, *SIGMOD Record*, v. 34, n. 3, pp. 49, 2005.
- [121] POWELL, W., OWEN-SMITH, J. “Universities and the Market for Intellectual Property in the Life Sciences”, *Journal of Policy Analysis and Management*, v. 17, n. 2, pp. 277, 1998.
- [122] MOREAU, L., MISSIER, P., BELHAJJAME, K., ET AL. *The PROV Data Model and Abstract Syntax Notation*. Relatório técnico, World Wide Web Consortium (W3C), dez. 2011. Disponível em: <http://www.w3.org/TR/prov-dm/>.
- [123] ALTSCHUL, S. F., GISH, W., MILLER, W., ET AL. “Basic local alignment search tool”, *Journal of Molecular Biology*, v. 215, n. 3, pp. 403–410, out. 1990. ISSN: 0022-2836. doi: 10.1016/S0022-2836(05)80360-2.
- [124] HENDERSON, R. “Job Scheduling Under the Portable Batch System”. In: *Job Scheduling Strategies for Parallel Processing - IPPS '95 Workshop*, v. 949, pp. 279–294. Springer, 1995.
- [125] FOSTER, I. “Globus Toolkit Version 4: Software for Service-Oriented Systems”. In: *IFIP International Conference on Network and Parallel Computing*, v. 3779, p. 13, 2006.
- [126] ANDERSON, R. *Security Engineering: A Guide to Building Dependable Distributed Systems*. 2008.
- [127] HOUSLEY, R., POLK, T. *Planning for PKI: Best Practices Guide for Deploying Public Key Infrastructure*. Wiley, 2001.

- [128] FOSTER, I., KESSELMAN, C., TSUDI, G., ET AL. “A security architecture for computational grids”. In: *Proceedings of the 5th ACM conference on Computer and communications security, CCS '98*, p. 83–92, New York, NY, USA, 1998. ACM. ISBN: 1-58113-007-4. doi: 10.1145/288090.288111. Disponível em: <<http://doi.acm.org/10.1145/288090.288111>>.
- [129] BUTLER, R., WELCH, V., ENDERT, D., ET AL. “A National-Scale Authentication Infrastructure”, *IEEE Computer*, v. 33, n. 12, pp. 66, 2000.
- [130] HABER, S., STORNETTA, W. S. “How to Time-Stamp a Digital Document”. In: Menezes, A. J., Vanstone, S. A. (Eds.), *Advances in Cryptology-CRYPT0' 90*, v. 537-455, Springer Berlin Heidelberg, pp. 437–455, Berlin, Heidelberg, 1990. ISBN: 978-3-540-54508-8. Disponível em: <<http://www.springerlink.com/content/89napu07upn1cp08/>>.
- [131] HABER, S., STORNETTA, W. “How to time-stamp a digital document”, *Journal of Cryptology*, v. 3, n. 2, pp. 99–111, 1991. ISSN: 0933-2790, 1432-1378. doi: 10.1007/BF00196791. Disponível em: <<http://www.springerlink.com/content/x771v2341pw02711/>>.
- [132] HABER, S., MASSIAS, H. “Time-stamping”. In: van Tilborg, H. (Ed.), *Encyclopedia of Cryptography and Security*, pp. 616–620. Springer, 2005.
- [133] CAO, B., PLALE, B., SUBRAMANIAN, G., ET AL. “Provenance Information Model of Karma Version 3”. In: *Proc. IEEE Congress on Services*, p. 351, 2009.
- [134] MYERS, J., FUTRELLE, J., PLUTCHAK, J., ET AL. “Embedding Data within Knowledge Spaces”, *CoRR*, v. abs/0902.0744, 2009.
- [135] ZHAO, Y., WILDE, M., FOSTER, I. “Applying the Virtual Data Provenance Model”. In: *Proc. 1st International Provenance and Annotation Workshop (IPAW 2006)*, v. 4145, *Lecture Notes in Computer Science*, pp. 148–161. Springer, 2006.
- [136] FURLANI, T., JONES, M., GALLO, S., ET AL. “Performance metrics and auditing framework for high performance computer systems”. In: *Proceedings of the 2011 TeraGrid Conference: Extreme Digital Discovery*, TG '11, p. 16:1–16:1. ACM, 2011. ISBN: 978-1-4503-0888-5. doi: 10.1145/2016741.2016759. Disponível em: <<http://doi.acm.org/10.1145/2016741.2016759>>.

- [137] “Technology Audit and Insertion Service for TeraGrid”. 2012. Disponível em: <http://www.si.umich.edu/research/project/technology-audit-and-insertion-service-teragrid>.
- [138] “XSEDE - Extreme Science and Engineering Discovery Environment”. 2012. Disponível em: <https://www.xsede.org>.
- [139] WIECZOREK, M., PRODAN, R., FAHRINGER, T. “Scheduling of scientific workflows in the ASKALON grid environment”, *SIGMOD Rec.*, v. 34, n. 3, pp. 56–62, set. 2005. ISSN: 0163-5808. doi: 10.1145/1084805.1084816. Disponível em: <http://doi.acm.org/10.1145/1084805.1084816>.
- [140] DUN, N., TAURA, K., YONEZAWA, A. “ParaTrac: a fine-grained profiler for data-intensive workflows”. In: *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, HPDC '10, pp. 37–48. ACM, 2010. ISBN: 978-1-60558-942-8. doi: 10.1145/1851476.1851482. Disponível em: <http://doi.acm.org/10.1145/1851476.1851482>.
- [141] LIEW, C., ATKINSON, M., OSTROWSKI, R., ET AL. “Performance database: capturing data for optimizing distributed streaming workflows”, *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, v. 369, n. 1949, pp. 3268–3284, 2011. doi: 10.1098/rsta.2011.0134.
- [142] ABITEBOUL, S., QUASS, D., MCHUGH, J., ET AL. “The Lorel query language for semistructured data”, *International Journal on Digital Libraries*, v. 1, pp. 66–88, 1997. Disponível em: <http://dx.doi.org/10.1007/s007990050005>.
- [143] CHEBOTKO, A., LU, S., FEI, X., ET AL. “RDFProv: A relational RDF store for querying and managing scientific workflow provenance”, *Data & Knowledge Engineering*, v. 69, n. 8, pp. 836–865, 2010. ISSN: 0169-023X. doi: 10.1016/j.datak.2010.03.005.
- [144] TONKIN, E., ALLINSON, J. “Signed Metadata: Method and Application”. In: *Proc. 2006 International Conference on Dublin Core and Metadata Applications*, p. 297, 2006.
- [145] BARTEL, M., BOYER, J., FOX, B., ET AL. “XML-Signature Syntax and Processing”. 2008. Disponível em: <http://www.w3.org/TR/xmlsig-core>.

- [146] TAN, V., GROTH, P., MILES, S., ET AL. “Security Issues in a SOA-Based Provenance System”. In: Hutchison, D., Kanade, T., Kittler, J., et al. (Eds.), *Provenance and Annotation of Data*, v. 4145, Springer Berlin Heidelberg, pp. 203–211, Berlin, Heidelberg, 2006. ISBN: 978-3-540-46302-3, 978-3-540-46303-0. Disponível em: <<http://www.springerlink.com/content/4nq166282j3g5842/>>.
- [147] BRAUN, U., SHINNAR, A. *A Security Model for Provenance*. Relatório Técnico TR-04-06, Computer Science Group, Harvard University, 2006.
- [148] MUNISWAMY-REDDY, K., HOLLAND, D., BRAUN, U., ET AL. “Provenance-Aware Storage Systems”. In: *Proc. USENIX Annual Technical Conference, General Track 2006*, p. 56, 2006.
- [149] BRAUN, U., SHINNAR, A., SELTZER, M. “Securing provenance”. In: *Proceedings of the 3rd conference on Hot topics in security*, p. 4:1–4:5, Berkeley, CA, USA, 2008. USENIX Association. Disponível em: <<http://dl.acm.org/citation.cfm?id=1496671.1496675>>.
- [150] HASAN, R., SION, R., WINSLETT, M. “Preventing history forgery with secure provenance”, *ACM Transactions on Storage*, v. 5, n. 4, pp. 12:1–12:43, dez. 2009. ISSN: 1553-3077. doi: 10.1145/1629080.1629082. Disponível em: <<http://doi.acm.org/10.1145/1629080.1629082>>.
- [151] DAI, C., LIN, D., BERTINO, E., ET AL. “An Approach to Evaluate Data Trustworthiness Based on Data Provenance”. In: *Proc. 5th VLDB Workshop on Secure Data Management (SDM 2008)*, v. 5159, pp. 98, Springer, 2008.
- [152] NAGAPPAN, M., VOUK, M. “A Model for Sharing of Confidential Provenance Information in a Query Based System”. In: *Proc. 2nd International Provenance and Annotation Workshop (IPAW 2008)*, v. 5272, pp. 69, Springer, 2008.
- [153] COOPER, D., SANTESSON, S., FARRELL, S., ET AL. “Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile”. 2008.
- [154] HASAN, R., SION, R., WINSLETT, M. “Introducing secure provenance: problems and challenges”. In: *Proceedings of the 2007 ACM workshop on Storage security and survivability*, StorageSS

- '07, p. 13–18, New York, NY, USA, 2007. ACM. ISBN: 978-1-59593-891-6. doi: 10.1145/1314313.1314318. Disponível em: <<http://doi.acm.org/10.1145/1314313.1314318>>.
- [155] MOREAU, L., ZHAO, Y., FOSTER, I., ET AL. “XDTM: XML Dataset Typing and Mapping for Specifying Datasets”. 2005.
- [156] JEWITT, D. “Project Pan-STARRS and the Outer Solar System”, *Earth, Moon, and Planets*, v. 92, n. 1, pp. 465–476, 2003. ISSN: 0167-9295. doi: 10.1023/B:MOON.0000031961.88202.60.
- [157] MATHOG, D. R. “Parallel BLAST on split databases”, *Bioinformatics*, v. 19, n. 14, pp. 1865–1866, 2003. doi: 10.1093/bioinformatics/btg250.
- [158] “Guidelines for Maintaining a Lab Notebook”. 2012.
- [159] HASAN, R., SION, R., WINSLETT, M. “The Case of the Fake Picasso: Preventing History Forgery with Secure Provenance”. In: *Proc. 7th USENIX Conference on File and Storage Technologies (FAST '09)*, p. 14, 2009.
- [160] XU, S., NI, Q., BERTINO, E., ET AL. “A Characterization of The Problem of Secure Provenance Management”. In: *Proc. IEEE International Conference on Intelligence and Security Informatics (ISI 2009)*, p. 314, 2009.
- [161] ALDECO-PÉREZ, R., MOREAU, L. “Securing Provenance-Based Audits”. In: McGuinness, D., Michaelis, J., Moreau, L. (Eds.), *Provenance and Annotation of Data and Processes*, v. 6378, *Lecture Notes in Computer Science*, Springer Berlin / Heidelberg, pp. 148–164, 2010. ISBN: 978-3-642-17818-4.
- [162] DAVIDSON, S. B., KHANNA, S., MILO, T., ET AL. “Provenance views for module privacy”. In: *Proceedings of the thirtieth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, PODS '11, p. 175–186. ACM, 2011. ISBN: 978-1-4503-0660-7. doi: 10.1145/1989284.1989305.
- [163] NI, Q., XU, S., BERTINO, E., ET AL. “An Access Control Language for a General Provenance Model”. In: Hutchison, D., Kanade, T., Kittler, J., et al. (Eds.), *Secure Data Management*, v. 5776, Springer, pp. 68–88, 2009. ISBN: 978-3-642-04218-8, 978-3-642-04219-5.
- [164] FOWLER, M. *Domain-Specific Languages*. Addison-Wesley, 2011.

- [165] “Sistema Nacional de Processamento de Alto Desempenho (SINAPAD)”. 2012. Disponível em: <<http://www.sinapad.lncc.br>>.
- [166] DEAN, J., GHEMAWAT, S. “MapReduce: simplified data processing on large clusters”. In: *Proceedings of the 6th conference on Symposium on Operating Systems Design & Implementation - Volume 6*, OSDI’04, p. 10–10, Berkeley, CA, USA, 2004. USENIX Association.
- [167] MELNIK, S., GUBAREV, A., LONG, J., ET AL. “Dremel: interactive analysis of web-scale datasets”, *Commun. ACM*, v. 54, n. 6, pp. 114–123, jun. 2011. ISSN: 0001-0782. doi: 10.1145/1953122.1953148. Disponível em: <<http://doi.acm.org/10.1145/1953122.1953148>>.
- [168] DONG, G., LIBKIN, L., SU, J., ET AL. “Maintaining the transitive closure of graphs in SQL”, *In Int. J. Information Technology*, v. 5, 1999.
- [169] “Provenance Challenge Wiki”. 2009. Disponível em: <<http://twiki.ipaw.info>>.
- [170] GROTH, P., MILES, S., MISSIER, P., ET AL. “A Proposal for Handling Collections in the Open Provenance Model”. 2009. Disponível em: <<http://mailman.ecs.soton.ac.uk/pipermail/provenance-challenge-ipaw-info/2009-June/000120.html>>.
- [171] CZAJKOWSKI, K., FOSTER, I., KARONIS, N., ET AL. “A Resource Management Architecture for Metacomputing Systems”. In: *Job Scheduling Strategies for Parallel Processing - IPPS/SPDP ’98 Workshop*, v. 1459, pp. 82, Springer, 1998.
- [172] WHITE, R., ROTH, R. *Exploratory Search: Beyond the Query–Response Paradigm*. Morgan & Claypool, 2009.
- [173] ORDONEZ, C. “Optimizing Recursive Queries in SQL”. In: *Proc. of the ACM SIGMOD International Conference on Management of Data (SIGMOD 2005)*, pp. 834–839, 2005.
- [174] DRIES, A., NIJSSEN, S. “Analyzing Graph Databases by Aggregate Queries”. In: *Proc. Workshop on Mining and Learning with Graphs (MLG 2010)*, pp. 37–45, 2010.
- [175] JAGADISH, H. V., CHAPMAN, A., ELKISS, A., ET AL. “Making database systems usable”. In: *Proceedings of the ACM SIGMOD International Conference on Management of Data*, p. 13–24. ACM, 2007.

ISBN: 978-1-59593-686-8. doi: 10.1145/1247480.1247483. Disponível em:
<<http://doi.acm.org/10.1145/1247480.1247483>>.

- [176] YU, C., JAGADISH, H. V. “Schema summarization”. In: *Proceedings of the 32nd international conference on Very large data bases, VLDB '06*, p. 319–330. VLDB Endowment, 2006. Disponível em:
<<http://dl.acm.org/citation.cfm?id=1182635.1164156>>.
- [177] ADAMS, C., CAIN, P., PINKAS, D., ET AL. “Internet X.509 Public Key Infrastructure Time-Stamp Protocol (TSP)”. 2001. Disponível em:
<<http://www.ietf.org/rfc/rfc3161.txt>>. IETF, RFC 3161.
- [178] HOUSLEY, R. “Cryptographic Message Syntax (CMS)”. 2004. Disponível em:
<<http://www.ietf.org/rfc/rfc3852.txt>>. IETF, RFC 3852.
- [179] CRUELLAS, J., KARLINGER, G., PINKAS, D., ET AL. “XML Advanced Electronic Signatures (XAdES)”. 2003. Disponível em:
<<http://www.w3.org/TR/XAdES>>.
- [180] “OpenSSL”. 2012. Disponível em: <<http://www.openssl.org>>.
- [181] “OpenTSA”. 2012. Disponível em: <<http://www.opentsa.org>>.
- [182] “XAdES add-on for XSECT”. 2012. Disponível em:
<<http://jce.iaik.tugraz.at/sic/Products/XML-Security/XAdES>>.

Apêndice A

Provenance Management in Many-Task Computing

A.1 Data Model

In Swift, data is represented by strongly-typed single-assignment variables. Data types can be *atomic* or *composite*. Atomic types are given by *primitive* types, such as integers or strings, or *mapped* types. Mapped types are used for representing and accessing data stored in local or remote files. *Composite* types are given by structures and arrays. In the Swift runtime, data is represented by a *dataset handle*. It may have as attributes a value, a filename, a child dataset handle (when it is a structure or an array), or a parent dataset handle (when it is contained in a structure or an array).

Swift processes are given by invocations of external programs, invocations of internal procedures, built-in functions, and operators. Dataset handles are produced and consumed by Swift processes.

In the Swift provenance model, dataset handles and processes are recorded, as are the relations between them (either a process consuming a dataset handle as input, or a process producing a dataset handle as output). Each dataset handle and process is uniquely identified in time and space by a URI. This information is stored persistently in a relational database. The two key relational tables used to

This appendix is based on the paper:

- L. Gadelha, B. Clifford, M. Mattoso, M. Wilde, and I. Foster. Provenance management in Swift. *Future Generation Computer Systems*, 27(6):775-780, Elsevier, 2011.

and a technical report containing implementation details:

- L. Gadelha, B. Clifford, M. Mattoso, M. Wilde, and I. Foster. Provenance Management in Swift with Implementation Details. Technical Report ANL/MCS-TM-311, Argonne National Laboratory, 2010.

Table A.1: Database relation `processes`.

Attribute	Definition
<code>id</code>	the URI identifying the process
<code>type</code>	the type of the process: execution, compound procedure, function, operator

Table A.2: Database relation `dataset_usage`.

Attribute	Definition
<code>process_id</code>	a URI identifying the process end of the relationship
<code>dataset_id</code>	a URI identifying the dataset handle end of the relationship
<code>direction</code>	whether the process is consuming or producing the dataset handle
<code>param_name</code>	the parameter name of this relation

store the structure of the provenance graph are `processes`, which stores brief information about processes (see table A.1), and `dataset_usage`, which stores produced and consumed relationships between processes and dataset handles (see table A.2). Other tables, illustrated in figure A.1, are used to record details about each process and dataset, and other relationships such as dataset containment. We do not give a detailed a description of the model in this section because it was a preliminary one that evolved to the one presented in detail in appendix B.

Consider the SwiftScript program in listing A.1, which first describes a procedure (`sortProg`, which calls the external executable `sort`); then declares references to two files, (`f`, a reference to `inputfile`, and `g`, a reference to `outputfile`); and finally calls the procedure `sortProg`. When this program is run, provenance records are generated as follows:

- a process record is generated for the initial call to the `sortProg(f)` procedure;

Listing A.1: SwiftScript program for sorting a file.

```

type file;
app (file o) sortProg(file i) {
    sort stdin=@filename(i) stdout=@filename(o);
}
file f <"inputfile">;
file g <"outputfile">;
g = sortProg(f);

```

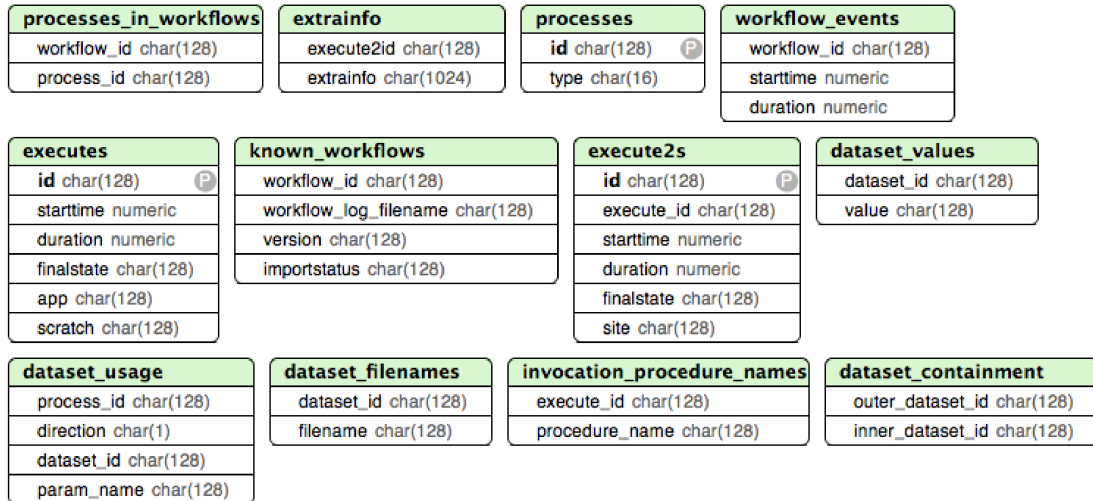


Figure A.1: Provenance database schema.

- a process record is generated for the `@filename(i)` function invocation inside `sortProg`, representing the evaluation of the `@filename` function that Swift uses to determine the physical filename corresponding to the reference `f`;
- and a process record is generated for the `@filename(o)` function invocation inside `sortProg`, again representing the evaluation of the `@filename` function, this time for the reference `g`.

Dataset handles are recorded for:

- the string `inputfile`;
- the string `outputfile`;
- the file variable `f`;
- the file variable `g`;
- the filename of `i`;
- and the filename of `o`.

Input and output relations are recorded as:

- the `sortProg(f)` procedure takes `f` as an input;
- the `sortProg(f)` procedure produces `g` as an output;
- the `@filename(i)` function takes `f` as an input;
- the `@filename(i)` function produces the filename of `f` as an output;

- the `@filename(o)` function takes `g` as an input;
- and the `@filename(o)` function produces the filename of `g` as an output.

The Swift provenance model is close to OPM, but there are some differences. Dataset handles correspond closely with OPM artifacts as immutable representations of data. However they do not correspond exactly, dataset handles do not record provenance due to aliasing, such as when accessing arrays. Section A.3 discusses this issue in more detail. The OPM entity “agent” was not represented explicitly in this version of Swift’s provenance model, however this can be represented, for instance, by the identity of the user that runs a workflow as an annotation in the model we propose in appendix B.

Except for *wasControlledBy*, the dependency relationships defined in OPM can be derived from the `dataset_usage` database relation. It explicitly stores the *used* and *wasGeneratedBy* relationships. *wasTriggeredBy* and *wasDerivedFrom* dependency relationships can also be inferred from `dataset_usage`, in the `sortProg` example we have, for instance, $f \xleftarrow{\text{wasDerivedFrom}} g$. Figure A.2 shows the provenance relationships captured by Swift’s provenance system for the `sortProg` example using OPM notation. Table A.3 shows the equivalence between tuples stored in the `dataset_usage` table and OPM relationships.

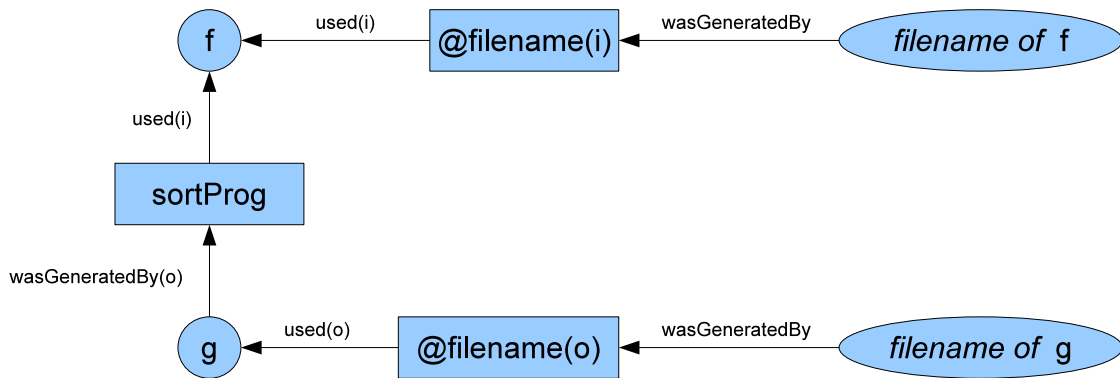
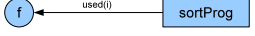

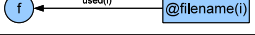
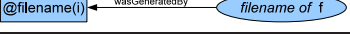
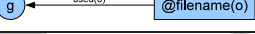
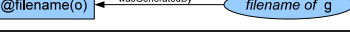


Figure A.2: Provenance relationships of a `sortProg` execution.

One of the main concerns with using a relational model for representing provenance is the need for querying over the transitive relation expressed in the `dataset_usage` table. For example, after executing the SwiftScript code in listing A.2, it might be desirable to find all dataset handles that lead to `c`: that is, `a` and `b`. However simple SQL queries over the `dataset_usage` relation can only go back one step, leading to the answer `b` but not to the answer `a`. To address this problem, we generate a transitive closure table by an incremental evaluation system [168]. This approach makes it straightforward to query over transitive relations using na-

Table A.3: Equivalence between tuples in the `dataset_usage` table and OPM.

Tuple in the <code>dataset_usage</code> table				OPM equivalent
<code>process_id</code>	<code>dataset_id</code>	<code>direction</code>	<code>param_name</code>	
<code>sortProg(f)</code>	<code>f</code>	In	<code>i</code>	
<code>sortProg(f)</code>	<code>g</code>	Out	<code>o</code>	
<code>@filename(i)</code>	<code>f</code>	In	<code>i</code>	
<code>@filename(i)</code>	filename of <code>f</code>	Out	<code>result</code>	
<code>@filename(o)</code>	<code>g</code>	In	<code>o</code>	
<code>@filename(i)</code>	filename of <code>g</code>	Out	<code>result</code>	

Listing A.2: Transitivity of provenance relationships.

```

b = p(a);
c = q(b);

```

tural SQL syntax, at the expense of larger database size and longer time to import provenance data from the log files.

Swift’s provenance data model is not dependent on a particular database model. A number of other forms were briefly experimented with during development. The two most developed and interesting models were XML and Prolog. XML provides a semi-structured tree form for data. A benefit of this approach is that new data can be added to the database without needing an explicit schema to be known to the database. In addition, when used with a query language such as XPath, certain transitive queries become straightforward with the use of the `//` operator of XPath. Representing the data as Prolog tuples is a different representation than a traditional database, but provides a query interface that can express interesting queries flexibly.

A.2 PC3: Implementation and Queries

One of the main goals of PC3 was to evaluate OPM as a mechanism for interoperability between provenance systems. An astronomy workflow from the Pan-STARRS [156] project, called LoadWorkflow, was used for this purpose. As illustrated in figure A.3, it receives a set of CSV files containing astronomical data, stores the contents of these files in a relational database, and performs a series of validation steps. This workflow makes extensive use of conditional and loop flow controls and database operations. Database operations are somewhat outside the scope of usual Swift applications, which are generally file-oriented. A Java implementation of the component applications of LoadWorkflow was provided in the Provenance Challenge Wiki [169]. These components are declared in the SwiftScript implementation of the workflow as external application procedures. The procedural body of the SwiftScript

code closely follows the LoadWorkflow specification since Swift has native support for decision and loop controls, given by the `if` and `foreach` constructs. A more detailed description of the activities can be found in tables A.4, A.4, and A.6.

In our initial attempts to implement LoadWorkflow, we found the use of the parallel `foreach` loop problematic because the database routines executed by the external application procedures are opaque to Swift. Due to dependencies between iterations of the loop, these routines were being incorrectly executed in parallel. It was necessary to serialize the loop execution to keep the database consistent. For the same reason, since most of the PC3 queries are for row-level database provenance, we had to implement a workaround for gathering this provenance by modifying the application database so that for every row inserted, an entry containing the execution identifier of the Swift process that performed this insertion is recorded on a separate annotation table.

The OPM output for a LoadWorkflow run in Swift was generated by a script that maps Swift's provenance data model to OPM's XML schema. Since OPM and Swift's provenance database use similar data models, it is fairly straightforward to build a tool to import data from an OPM graph into the Swift provenance database. However we observed that the OPM outputs from the various participating teams, including Swift, carry many details of the LoadWorkflow implementation that are system specific, such as auxiliary tasks that are not necessarily related to the workflow. To answer the same queries, it would be necessary to perform some manual interpretation of the imported OPM graph in order to identify the relevant processes and artifacts.

A detailed description of the LoadWorkflow implementation in SwiftScript, and the SQL queries to the provenance database can be found in [91]. Next we describe some of the queries performed:

Core Query 1. The first query asks, for a given detection, which CSV files contributed to it. The strategy used to answer this query is to determine input CSV files that precede, in the transitivity table, the process that inserted the detection. Suppose we want to determine the provenance of the detection that has the identifier 261887481030000003, the first query can be answered by first obtaining the Swift process identifier of the process that inserted the detection from the annotations included in the application database:

```
> select
  provenanceid
from
  ipaw.p2detectionprov
where
  detectid = 261887481030000003;

> execute2:pc3-20090507-1140-z7ebbrz0:ps_load_executable_db_app-8d52pga
```

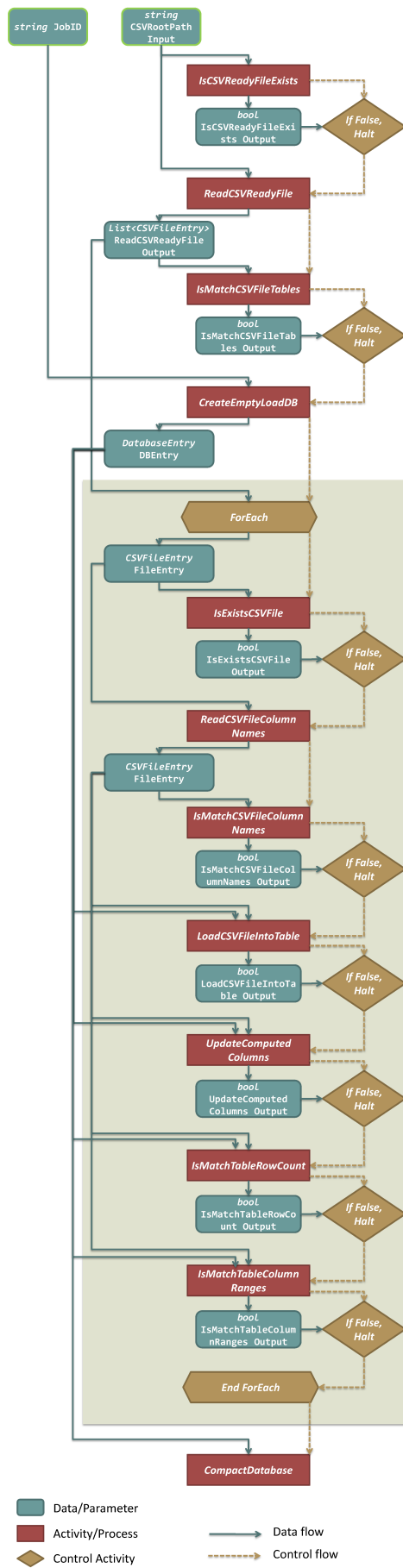


Figure A.3: LoadWorkflow. Source: PC3 web site [5].

Activity	Input	Output
IsCSVReadyFileExists: Verifies if the CSV root directory and the <code>csv_ready.csv</code> file exist.	string CSVRootPathInput, containing the path to the CSV root directory.	boolean IsCSVReadyFileExistsOutput, which is true if the verification succeeds, or false otherwise.
ReadCSVReadyFile: For each file listed in <code>csv_ready.csv</code> , it creates a CSVFileEntry, which consists of the path to the CSV file to be loaded, the path to the CSV header file containing the list of data columns, the number of rows in the file, the target database table, and the MD5 hash function value of the file. The columns names field is not populated by this activity.	string CSVRootPathInput, containing the path to the CSV root directory.	list ReadCSVReadyFileOutput of CSVFileEntry elements read from <code>csv_ready.csv</code> .
IsMatchCSVFileTable: Verifies if the tables to be loaded have corresponding data files.	list FileEntriesInput of CSVFileEntry elements read from <code>csv_ready.csv</code> .	boolean IsMatchCSVFileTablesOutput, which is true if the tables have corresponding CSV files, or false otherwise.
IsExistsCSVFileTable: Verifies if CSV data file and CSV header exist.	CSVFileEntry FileEntryInput.	boolean IsExistsCSVFileOutput, which is true if the CSV data file and CSV header files exist, or false otherwise.
ReadCSVFileColumnNames: Reads the list of column names present in the CSV data file from the CSV header file.	CSVFileEntry FileEntryInput.	CSVFileEntry FileEntryOutput, which results from updating the columns names field in the input using the values listed in the CSV header file.
IsMatchCSVFileColumnNames: Verifies if the columns expected for a target table are present in the CSV data file.	CSVFileEntry FileEntryInput, with the columns names field populated.	boolean IsMatchCSVFileColumnNamesOutput, which is true if column names listed in the CSV data files match the column names for the target database table, or false otherwise.

Table A.4: LoadWorkflow Activities - Pre-Load Section

Activity	Input	Output
CreateEmptyLoadDB: Creates the database to which the CSV data files will be loaded. It returns a DatabaseEntry, which is a reference to the database containing its name and connection information.	string JobID, a unique job identifier for the batch of CSV data files.	A DatabaseEntry CreateEmptyLoadDBOutput.
LoadCSVFileIntoTable: Loads a CSV data file into the corresponding database table.	DatabaseEntry DBEntry, containing target table to load the CSV data file into. CSVFileEntry FileEntry, referring to the CSV data file to be loaded.	boolean LoadCSVFileIntoTableOutput, which is true if the load was successful, or false otherwise.
UpdateComputedColumns: Updates the computed columns in the table that was loaded. These columns are indicated by the value -999 in the CSV data file.	DatabaseEntry DBEntry, with the target table already loaded from the CSV data file. CSVFileEntry FileEntry, containing the name of target table in the database to update.	boolean UpdateComputedColumnsOutput, which is true if the columns were successfully updated, or false otherwise.

Table A.5: LoadWorkflow Activities - Load Section

Activity	Input	Output
IsMatchTableRowCount: Checks if number of rows loaded into table matches the expected.	DatabaseEntry DBEntry, where the target table is loaded and updated. CSVFileEntry FileEntry, containing the expected number of rows in the CSV data file and the target database table name.	bool IsMatchTableRowCountOutput, which is true if the number of rows in the target table matches the expected number of rows in the CSV data file.
IsMatchTableColumnRanges: Checks if the data loaded into database table columns is within the range of values expected.	DatabaseEntry DBEntry, where the target table is loaded and updated. CSVFileEntry FileEntry, containing the name of target table in the database to validate columns ranges.	bool IsMatchTableColumnRangesOutput, which is true if the data values of the columns in the target table are within the expected range, or false otherwise.
CompactDatabase: Compacts the database before concluding the workflow.	DatabaseEntry DBEntry, where all tables are loaded and validated.	None.

Table A.6: LoadWorkflow Activities - Post-Load Section

The identifier returned is an `execute2` identifier, which means in this case that it refers to a successful execution attempt. In order to obtain the predecessors of this process in the transitivity table we need the actual execute identifier of the process, which we can get with the following SQL query:

```
> select
  execute_id
from
  execute2s
where
  id = 'execute2:pc3-20090507-1140-z7ebbrz0:ps_load_executable_db_app-8d52pgaj';

> execute:pc3-20090507-1140-z7ebbrz0:0-5-5-1-5-1-2-0
```

Finally, we determine the filenames of datasets that contain CSV inputs in the set of predecessors of the process that inserted the detection:

```
> select
  filename
from
  trans, dataset_filenames
where
  after='execute
      :pc3-20090507-1140-z7ebbrz0:0-5-5-1-5-1-2-0'
and
  before=dataset_id and filename like '%split%';

> file://split_list_output-65fe229c-2da2-4054-997e-fb167b8c30ed--array/elt-3
file://split_list_output-65fe229c-2da2-4054-997e-fb167b8c30ed--array/elt-2
file://split_list_output-65fe229c-2da2-4054-997e-fb167b8c30ed--array/elt-1
```

These files contain the filenames of the CSV files that were given as input to the workflow, and that will result in the detection row insertion:

```
P2_J062941_B001_P2fits0_20081115_P2Detection.csv,
P2_J062941_B001_P2fits0_20081115_P2ImageMeta.csv,
P2_J062941_B001_P2fits0_20081115_P2FrameMeta.csv
```

Core Query 2. The second query asks if the range check (`IsMatchColumnRanges`) was performed in a particular table, given that a user found values that were not expected in it. This is implemented in the `q2.sh` script in the Swift SVN repository with the following SQL query:

```
> select
  dataset_values.value
from
  processes, invocation_procedure_names, dataset_usage, dataset_values
where
  type='compound' and
  procedure_name='is_match_table_column_ranges' and
  dataset_usage.direction='0' and
  dataset_usage.param_name='inputcontent' and
  processes.id = invocation_procedure_names.execute_id and
  dataset_usage.process_id = processes.id and
  dataset_usage.dataset_id = dataset_values.dataset_id;
```

This returns the input parameter XML for all `IsMatchColumnRanges` calls. These are XML values, and it is necessary to examine the resulting XML to determine if it was invoked for the specific table. There is unpleasant cross-format joining necessary here to get an actual yes/no result properly, although probably could use a `LIKE` clause to peek inside the value.

Core Query 3. The third core query asks which operation executions were strictly necessary for the `Image` table to contain a particular (non-computed) value. This uses the additional annotations made, that only store which process originally inserted a row, not which processes have modified a row. So to some extent, rows are regarded a bit like artifacts (though not first order artifacts in the provenance database); and we can only answer questions about the provenance of rows, not the individual fields within those rows. That is sufficient for this query, though. First find the row that contains the interesting value and extract its `IMAGEID`. Then find the process that created the `IMAGEID` by querying the Derby database table `P2IMAGEPROV`:

```
> select * from ipaw.p2imageprov where imageid=6294301;

IMAGEID | PROVENANCEID
-----
6294301 | execute2:pc3-20090519-2057d8dyi9o9:ps_load_executable_db_app-dpc8q1bj
```

Now we have a process ID for the process that created the row. Now query the transitive closure table for all predecessors for that process (as in the first core query). This will produce all processes and artifacts that preceded this row creation. Our answer differs from the sample answer because we have sequenced access to the database, rather than regarding each row as a proper first-order artifact. The entire database state at a particular time is a successor to all previous database accessing operations, so any process which led to any database access before the row in question is regarded as a necessary operations. This is undesirable in some respects, but desirable in others. For example, a row insert only works because previous database operations which inserted other rows did not insert a conflicting primary key - so there is data dependency between the different operations even though they operate on different rows.

Optional Query 1. The workflow halts due to failing an `IsMatchTableColumnRanges` check. How many tables successfully loaded before the workflow halted due to a failed check? This counts how many load processes are known to the database (over all recorded workflows):

```
> select
  count(*)
from
  invocation_procedure_names
where
  procedure_name='load_csv_file_into_table';
```

This can be restricted to a particular workflow run like this:

```
> select
  count(process_id)
from
  invocation_procedure_names, processes_in_workflows
where
  procedure_name='load_csv_file_into_table'
and
  workflow_id='execute:pc3-20090519-1659-jqc5od2f:run'
and
  invocation_procedure_names.execute_id = processes_in_workflows.process_id;

> 3
```

Optional Query 2. Which pairs of procedures in the workflow could be swapped and the same result still be obtained (given the particular data input)? In our Swift representation of the workflow, we control dataflow dependencies. So many of the activities that could be commuted are in our implementation run in parallel. One significant thing one cannot describe in SwiftScript (and so cannot answer from the provenance database using this method) is commuting operations on the database. From a Swift perspective, this is a limitation of our SwiftScript language rather than in the provenance implementation. The query lists which pairs Unix process executions (of which there are 50x50) have no data dependencies on each other. There are 2082 rows. The base SQL query is this:

```
> select
  L.id, R.id
from
  processes as L, processes as R
where
  L.type='execute'
and
  R.type='execute'
and
  NOT EXISTS (select * from trans where before=L.id and after=R.id);
```

This answer is deficient in a few ways. We do not take into account non-execute procedures (such as compound procedures, function invocations, and operator executions) - there are 253 processes in total, 50 being executes and the remainder being the other kinds of process. If we did that naively, we would not take into account compound procedures which contain other procedures (due to lack of decent support for nested processes - something like OPM accounts) and would come up with commutations which do not make sense.

A.3 PC3: Evaluation

PC3 provided an opportunity to use OPM in practice. This also enabled us to evaluate OPM and compare it to Swift's provenance data model. OPM originally

Listing A.3: Multiple provenance descriptions for a dataset.

```
int a = 7;  
int b = 10;  
int c[] = [a, b];
```

did not specify a naming mechanism for globally identifying artifacts outside of an OPM graph. In Swift, dataset handles are given an URI, now OPM has an annotation for this purpose [51].

Swift’s provenance implementation has two models of representing containment for dataset handles contained inside other dataset handles (arrays and complex types). A constructor/accessor model has special processes called accessors and constructors corresponding to the `[]` array accessor and `[1,2,3]` explicit construction syntax in SwiftScript. This model is proposed in OPM. In the Swift implementation, this is a cause of multiple provenances for dataset handles. For example, consider the SwiftScript program displayed in listing A.3, the expression `c[0]` evaluates to the dataset handle corresponding to the variable `a`. That dataset handle has a provenance trace indicating it was assigned from the constant value 7. However, that dataset handle has additional provenance indicating that it was output by applying the array access operator `[]` to the array `c` and the numerical value 0. In OPM, the artifact resulting from evaluating `c[0]` is distinct from the artifact resulting from evaluating `a`, although they may be annotated with an *isIdenticalTo* arc [170]. In order to address the divergence between OPM and Swift’s provenance data model, the dataset handle implementation could be modified so that it supported dataset handles being aliases to other dataset handles. The alias dataset handle would behave identically to the dataset handle that it aliases, except that it would have different provenance reflecting both the provenance of the original dataset handle, and subsequent operations made to retrieve it. In listing A.3, then, `c[0]` would return a newly created dataset handle that aliased the original dataset handle for `a`. There is also a container/contained model, where relations are stored directly between dataset handles indicating that one is contained inside the other, without intervening processes. These relations can be inferred from the constructor/accessor model. The *Contained* relation between two artifacts, defined in [170], indicates that one is contained within another. This maps relatively cleanly to Swift’s in-memory model of dataset handles containing other dataset handles. Swift collections may be hierarchical. In [170], it is not specified if the *Contained* relation holds only one level deep or to all elements contained in a collection.

The Swift team made a proposal [169] for a minor change to the XML schema to better reflect the perceived intentions of the OPM authors. It was apparent that the present representation of hierarchical processes in OPM is insufficiently rich for some

groups and that it would be useful to represent hierarchy of individual processes and their containing processes more directly. In Swift this is given by two categories: at the highest level, SwiftScript language constructs, such as procedures and functions; below that, the mechanics of Swift's execution, such as moving files to and from computational resources, and interactions with job execution. Swift provenance work to date has concentrated on the high-level representation, treating all of the low-level behavior as opaque and exposing neither processes nor artifacts. An OPM modification proposal for this is forthcoming. In Swift, this information is often available through the Karajan [171] execution engine thread identifier which closely maps to the Swift process execution hierarchy: a Swift process contains another Swift process if its Karajan thread identifier is a prefix of the second processes Karajan thread identifier. The Swift provenance database stores values of dataset handles when those values exist in-memory (for example, when a dataset handle represents an integer or a string). During PC3, interest in a standard way to represent this was expressed.

Apêndice B

MTCProv: A Practical Provenance Query Framework for Many-Task Scientific Computing

B.1 Provenance Query Patterns

In the context of many-task scientific computations, provenance queries can be application-independent, where the user is usually interested in information that is present in every run of a Swift script, such as production and consumption relationships between data sets and processes, and data set containment relationships. We identified the following patterns for application-independent provenance queries:

Entity Attribute (EA). Queries for attributes of an entity of the data model. Some of these queries may also be application-specific, as queries for the contents of variables that store values of atomic in-memory data sets given by scientific parameters of an application.

One-step Relationship (R). Queries for entities involved in a relationship of the data model, such as data set consumption and production, or data set containment.

Multiple-step Relationship (R)*. Queries for entities involved in the transitive closure of a relationship of the data model, e.g. for data set lineage and for data set containment hierarchy.

This appendix is based on the papers:

- L. Gadelha, M. Mattoso, M. Wilde, and I. Foster, Provenance Query Patterns for Many-Task Scientific Computations. *Proceedings of the 3rd USENIX Workshop on Theory and Applications of Provenance (TaPP'11)*, USENIX, 2011.
- L. Gadelha, M. Wilde, M. Mattoso, and I. Foster. Exploring Provenance in High Performance Scientific Computing. *Proceedings of the 1st Workshop on High-Performance Computing meets Databases (HPCDB 2011)*, p. 17-20. Co-located with Supercomputing (SC 2011), Seattle, USA. ACM, 2011.
- L. Gadelha, M. Wilde, M. Mattoso, and I. Foster, MTCProv: a practical provenance query framework for many-task scientific computing. *Distributed and Parallel Databases*, 30(5-6):351-370. Springer, 2012.

Lineage Graph Matching (LGM). Queries for determining similarity between lineage graphs. It can include a combination of EA, R, and R* queries, along with graph similarity algorithms to verify, for instance, common subgraphs, and graph difference. These problems are known to have high computational complexity in general.

One can observe that each application-independent query pattern is a generalization of the preceding one. Application-independent queries can be used to describe the structure of production and consumption provenance relationships. However, to understand and analyze the results of a computational experiment, a scientist often needs information that goes beyond structural provenance. Some query patterns depends more on which specific scientific computation was executed, since they involve queries about inherent attributes of a scientific application, such as values for input and output scientific parameters:

Run summary (RS). Given by application specific attributes of a script run or of the entities it contains (all its processes and their respective input/output data sets). Two special cases of this pattern are frequently found:

- *Run resource-level performance* (RRP). Given by information about the runtime behavior of the application executions of the script run. Some of this information is available as attributes of the **process** entity, therefore this pattern can be considered a special case of the EA pattern. In Swift, one can optionally monitor and record resource usage statistics such as memory allocation and the amount of data read or written to the file system by an application execution.
- *Run science-level performance* (RSP). Given by queries for input and output scientific parameters. These are gathered either as entity attributes, when given by scientific parameters that are eventually stored in in-memory variables during a script run; or as annotations, when this information is not directly visible to Swift. Such queries would be typified by questions such as “how did my runs perform on some key scientific indicator, for example, the energy level or size of a predicted protein structure?”.

Run comparisons (RCp). These queries compare multiple script runs with respect to some attribute (scientific or runtime, for instance) to analyze how it varied across them. One might want to know what was the range of versions of a particular application, or which protein was modeled in a bioinformatics workflow, across multiple script runs.

Run correlations (RCr). Given by queries for correlating attributes from multiple script runs. One can, for instance, correlate the resulting accuracy of some com-

putational model run (science-level performance) with the duration of its execution (resource-level performance).

Campaign-level summaries (CLS). These queries aggregate data from a *campaign*, i.e. a set of runs of a script related to a particular event or study. Some examples of this pattern would be querying for average computational model accuracy, or for total number of computational tasks in a campaign.

Table B.1 describes which patterns are present in each query proposed in the Provenance Challenge series. One can observe that some patterns, such as LGM and RCr, are present in only a few queries of the Provenance Challenges. However some useful queries can be designed exploring these patterns. Users of our provenance management system, from different scientific domains, were frequently interested in queries that present application-specific patterns, in particular for correlations between scientific-level and resource-level performance attributes.

Table B.1: Provenance query patterns found in the Provenance Challenge series.

Pattern	PC1/PC2									PC3				PC3 (Optional Queries)														
	1	2	3	4	5	6	7	8	9	1	2	3	5	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
EA	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
R	x	x	x		x	x	x	x	x	x	x	x	x		x	x	x	x	x	x	x	x	x	x	x	x	x	x
R*	x	x	x			x	x				x	x	x	x		x		x			x	x			x	x	x	x
LGM							x									x												
RS	x	x	x							x	x	x	x	x		x	x						x	x	x	x	x	
RCp					x	x	x	x	x							x			x	x	x	x						x
RCr				x																								

The Open Protein Simulator (OOPS) [97] is a protein structure prediction application. It is used in conjunction with pre-processing and post-processing applications in a high-level workflow that is described in figure B.1. The doLoopRound workflow activity is a compound procedure, also described in figure B.1. Annotations are gathered by an application-specific script executed after an OOPS run, and stored in the provenance database.

Scientists were interested to know, for instance, what the correlation between *root mean square distance* (RMSD), which measures how similar the modeled protein is to the actual one, and the number of simulation (loopModel) steps was for a given protein. This type of query, which presents the RCr pattern, can help the scientist to estimate the required number of simulation steps for achieving the desired accuracy. This is useful since extra simulation steps are usually computationally expensive. The following query returns the desired answer for the protein TR567:

```
SELECT run_id, r.value as nSim, t.value as rmsd
FROM   compare_run_by_param('proteinId') as r
       INNER JOIN
       compare_run_by_param('nSim') as s USING (run_id)
       INNER JOIN
```

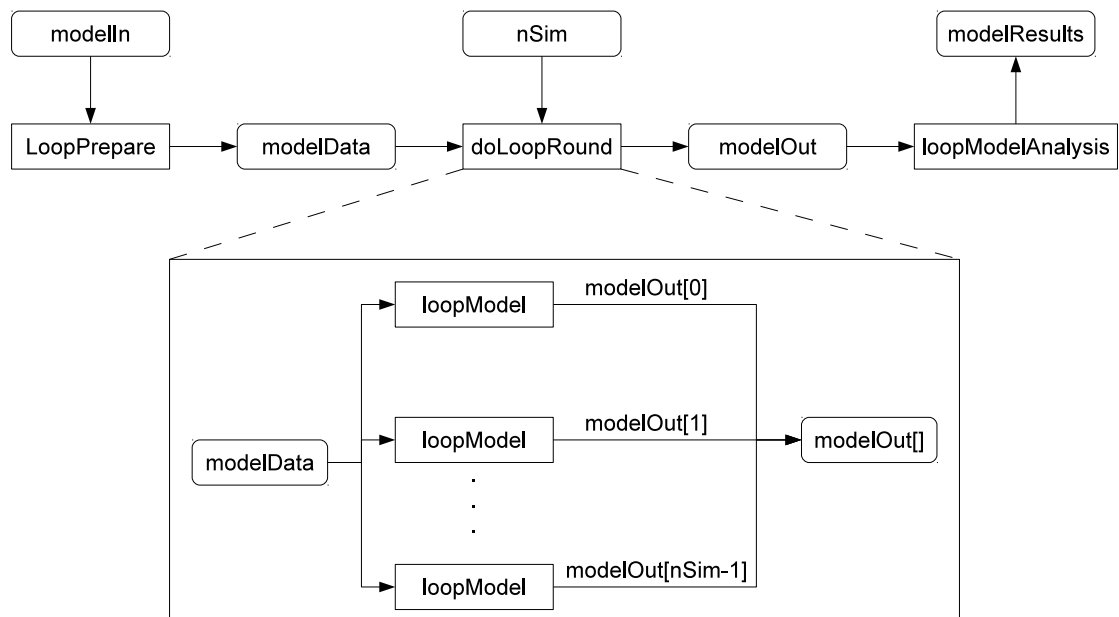


Figure B.1: Protein structure prediction workflow.

```

compare_run_by_annot('rmsd') as t USING (run_id)
WHERE r.value='TR567' and run.id LIKE 'psim.loops%';
  
```

run_id	nSim	rmsd
psim.loops-20100604-2215-cdifsnb3	256	3.33123
psim.loops-20100613-0125-keyyyc35	512	0.76274
psim.loops-20100616-1512-h6q4g4ja	1024	0.68426
...		

Where `compare_run_by_param` and `compare_run_by_annot` are functions that abstract the RCp pattern. They compare how the value of a parameter and the value associated to an annotation key varies across different runs respectively. The composition of these functions allows for the design of queries that present the RCr pattern. Queries of the R* pattern are supported by the SQL function `ancestors`, that uses Common Table Expressions to define a recursive query.

```

CREATE OR REPLACE FUNCTION ancestors(vvarchar)
RETURNS SETOF varchar AS $$
WITH RECURSIVE anc(ancestor,descendant) AS
(
  SELECT parent AS ancestor, child AS descendant
  FROM prov_graph
  WHERE child=$1
  UNION
  SELECT prov_graph.parent AS ancestor,
         anc.descendant AS descendant
  FROM anc, prov_graph
  WHERE anc.ancestor=prov_graph.child
)
  
```

```
SELECT ancestor FROM anc
$$ LANGUAGE SQL;
```

Where `prov_graph` is a database view that defines the edges of the provenance graphs stored in the database. An invocation of this procedure returns:

```
SELECT *
FROM   ancestors('dataset:20100618-0402-ia0bqb73:72000045');

          ancestor
-----
execute:psim.loops-20100618-0402-qhm9ugg4:451006
dataset:20100618-0402-ia0bqb73:72000039
...
```

The EA, R and RS query patterns are well supported using native SQL. Support for LGM queries is planned as work.

B.2 An Extended Provenance Model for Many-Task Scientific Computations

In this section, we present the MTCProv data model for representing provenance of many-task scientific computations. This MTC provenance model is a compatible extension of the Open Provenance Model, in the sense that it is possible to export the data stored by MTCProv to an OPM-compliant graph. It addresses the characteristics of many-task computation, where concurrent component tasks are submitted to parallel and distributed computational resources. Such resources are subject to failures, and are usually under high demand for executing tasks and transferring data. Science-level performance information, which describes the behavior of an experiment from the point of view of the scientific domain, is critical for the management of such experiments (for instance, by determining how accurate the outcome of a scientific simulation was, and whether accuracy varies between execution environments). Recording the resource-level performance of such workloads can also assist scientists in managing the life cycle of their computational experiments. In designing MTCProv, we interacted with Swift users from multiple scientific domains, including protein science, and earth sciences, and social network analysis, to support them in designing, executing and analyzing their scientific computations with Swift. From these engagements, we identified the following requirements for MTCProv:

1. *Gather producer-consumer relationships between data sets and processes.* These relationships form the core of provenance information. They enable typical provenance queries to be performed, such as determining all processes and data sets that were involved in the production of a particular data

set. This in general requires traversing a graph defined by these relationships. Users should be able, for instance, to check the usage of a given file by different many-task application runs.

2. *Gather hierarchical relationships between data sets.* Swift supports hierarchical data sets, such as arrays and structures. For instance, a user can map input files stored in a given directory to an array, and later process these files in parallel using a `foreach` construct. Fine-grained recording of data set usage details should be supported, so that a user can trace, for instance, that an array was passed to a procedure, and that an individual array member was used by some sub-procedure. This is usually achieved by recording constructors and accessors of arrays as processes in a provenance trace [51].
3. *Gather versioned information of the specifications of many-task scientific computations and of their component applications.* As the specifications of many-task computations (e.g., Swift scripts), and their component applications (e.g., Swift leaf functions) can evolve over time, scientists can benefit from keeping track of which version they are using in a given run. In some cases, the scientist also acts as the developer of a component application, which can result in frequent component application version updates during a workflow lifecycle.
4. *Allow users to enrich their provenance records with annotations.* Annotations are usually specified as key-value pairs that can be used, for instance, to record resource-level and science-level performance, like input and output scientific parameters, and usage statistics from computational resources. Annotations are useful for extending the provenance data model when required information is not captured in the standard system data flow. For instance, many scientific applications use textual configuration files to specify the parameters of a simulation. Yet automated provenance management systems usually record only the consumption of the configuration file by the scientific application, but preserve no information about its content (which is what the scientist really needs to know).
5. *Gather runtime information about component application executions.* Systems like Swift support many diverse parallel and distributed environments. Depending on the available applications at each site and on job scheduling heuristics, a computational task can be executed on a local host, a high performance computing cluster, or a remote grid or cloud site. Scientists often can benefit from having access to details about these executions, such as where each job was executed, the amount of time a job had to wait on a scheduler's queue, the

duration of its actual execution, its memory and processor consumption, and the volume and rate of file system and/or network IO operations.

6. *Provide a usable and useful query interface for provenance information.* While the relational model is ideal for many aspects to store provenance relationships, it is often cumbersome to write SQL queries that require joining the many relations required to implement the OPM. For provenance to become a standard part of the e-Science methodology, it must be easy for scientists to formulate queries and interpret their outputs. Queries can often be of exploratory nature [172], where provenance information is analyzed in many steps that refine previous query outputs. Improving usability is usually achievable through the specification of a provenance query language, or by making available a set of stored procedures that abstract common provenance queries. *In this work we propose a query interface which both extends and simplifies standard SQL by automating join specifications and abstracting common provenance query patterns into built-in functions.*

Some of these requirements have been previously identified by Miles et al. [71]. However few works to date emphasize usability and applications of provenance represented by requirements 4 through 6, which are the main objectives of this work. As a first step toward meeting these requirements, we propose a data model for the provenance of many-task scientific computations. A UML diagram of this provenance model is presented in Figure B.2. We simplify the UML notation to abbreviate the information that each annotated entity set (script run, function call, and variable) has one annotation entity set per data type. We define entities that correspond to the OPM notions of artifact, process, and artifact usage (either being consumed or produced by a process). These are augmented with entities used to represent many-task scientific computations, and to allow for entity annotations. Such annotations, which can be added post-execution, represent information about provenance entities such as object version tags and scientific parameters.

ScriptRun refers to the execution (successful or unsuccessful) of an entire many-task scientific computation, which in Swift is specified as the execution of a complete parallel script from start to finish. *FunctionCall* records calls to Swift functions. These calls take as input data sets, such as values stored in primitive variables or files referenced by mapped variables; perform some computation specified in the respective function declaration; and produce data sets as output. In Swift, function calls can represent invocations of external applications, built-in functions, and operators; each function call is associated with the script run that invoked it. *ApplicationFunctionCall* represents an invocation of one component application of a many-task scientific computation. In Swift, it is generated by an invocation to an

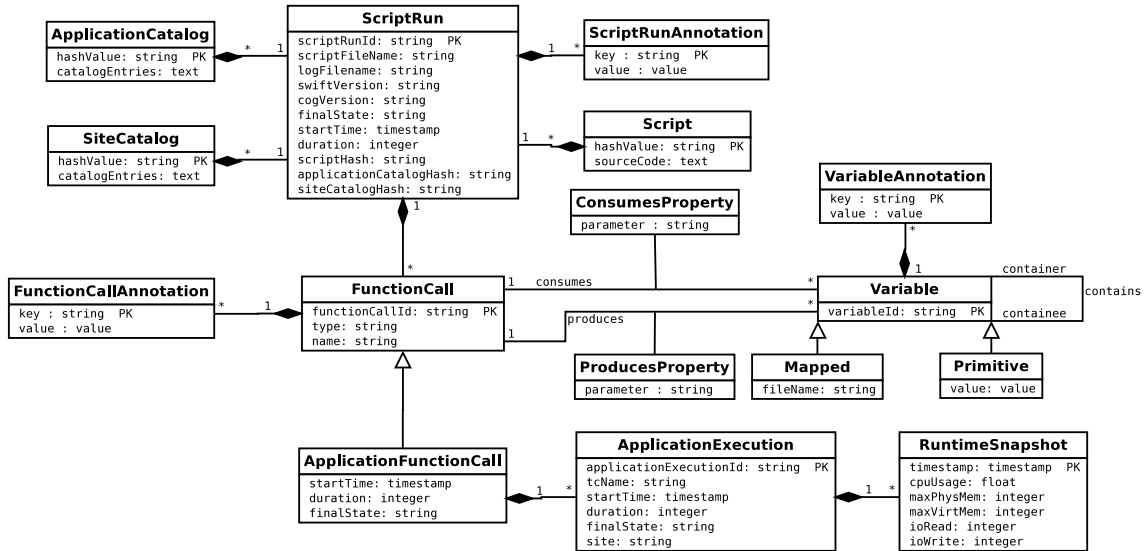


Figure B.2: UML diagram of Swift’s provenance database.

external application. External applications are listed in an application catalog along with the computational resources on which they can be executed. *ApplicationExecution* represents execution attempts of an external application. Each application function call triggers one or more execution attempts, where one (or, in the case of retries or replication, several) particular computational resource(s) will be selected to actually execute the application. *RuntimeSnapshot* contains information associated with an application execution, such as resource consumption. *Variable* represents data sets that were assigned to variables in a Swift script. Variable types can be atomic or composite. Atomic types are primitive types, such as integers and strings, recorded in the relation *Primitive*, or *mapped* types, recorded in the relation *Mapped*. Mapped types are used for declaring and accessing data that is stored in files. Composite types are given by structures and arrays. Containment relationships define a hierarchy where each variable may have child variables (when it is a structure or an array), or a parent variable (when it is a member of a collection). A variable may have as attributes a value, when it is a primitive variable; or a filename, when it is a mapped file. An *(Entity set name)Annotation* is a key-value pair associated with either a variable, function call, or script run. These annotations are used to store context-specific information about the entities of the provenance data model, as illustrated in figure B.3. Examples include scientific-domain parameters, object versions, and user identities. Annotations can also be used to associate a set of runs of a script related to a particular event or study, which we refer to as a *campaign*. The *produces* and *consumes* relationships between *FunctionCall* and *Variable* define a lineage graph that can be traversed to determine ancestors or descendants of a particular entity. Process dependency and data dependency graphs are derived with

transitive queries over these relationships.

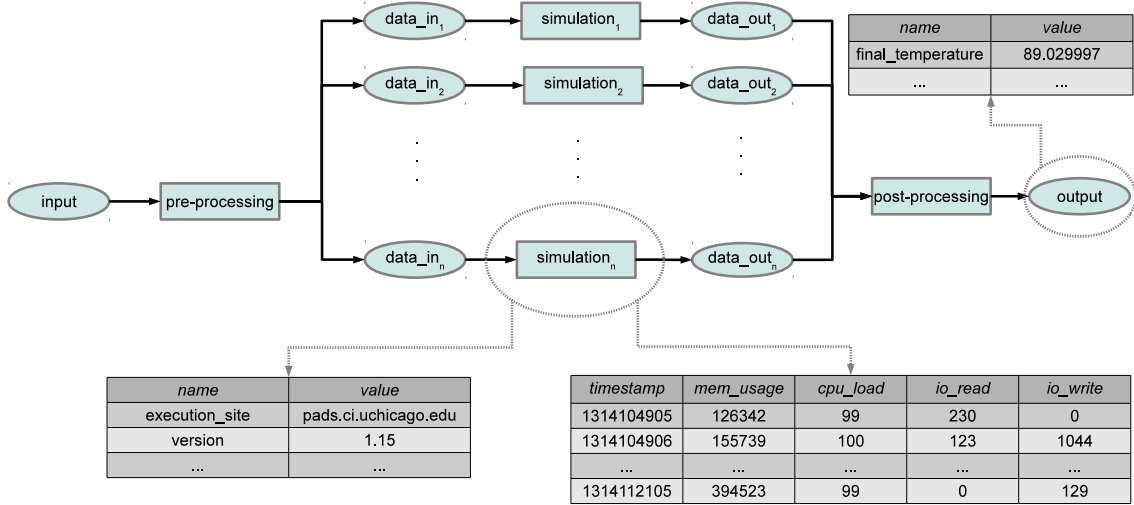


Figure B.3: Examples of annotations enabled by MTCProv.

The provenance model presented here is a significant refinement of a previous one used by Swift in the Third Provenance Challenge [64], which was shown to be similar to OPM. This similarity is retained in the current version of the model, which adds support for annotations and runtime information on component application executions. *FunctionCall* corresponds to OPM processes, and *Variable* corresponds to OPM artifacts as immutable data objects. The OPM entity agent controls OPM processes, e.g. starting or terminating them. While we do not explicitly define an entity type for such agents, this information can be stored in the annotation tables of the *FunctionCall* or *ScriptRun* entities. To record which users controlled each script or function call execution, one can gather the associated POSIX userids, when executions are local, or the distinguished name of network security credentials, when executions cross security domains, and store them as annotations for the respective entity. This is equivalent to establishing an OPM *wasControlledBy* relationship. The dependency relationships, *used* and *wasGeneratedBy*, as defined in OPM, correspond to our *consumes* and *produces* relationships, respectively. Our data model has additional entity sets to capture behavior that is specific to parallel and distributed systems, to distinguish, for instance, between application invocations and execution attempts. We currently do not directly support the OPM concept of *account*, which can describe the same computation using different levels of abstraction. However, one could use annotations to associate one or more such accounts with an execution entity. Based on the mapping to OPM described here, MTCProv provides tools for exporting the provenance database into OPM provenance graph interchange format, which provides interoperability with other OPM-compatible provenance systems.

B.3 Design and Implementation of MTCProv

In this section, we describe the design and implementation of the MTCProv provenance query framework. It consists of a set of tools used for extracting provenance information from Swift log files, and a query interface. While its log extractor is specific to Swift, the remainder of the system, including the query interface, is applicable to any parallel functional data flow execution model. The MTCProv system design is influenced by our survey of provenance queries in many-task computing [92], where a set of query patterns was identified. The *multiple-step relationships* (R^*) pattern is implemented by queries that follow the transitive closure of basic provenance relationships, such as data containment hierarchies, and data derivation and consumption. The *run correlation* (RCr) pattern is implemented by queries for correlating attributes from multiple script runs, such as annotation values or the values of function call parameters.

B.3.1 Provenance Gathering and Storage

Swift can be configured to add both prospective and retrospective provenance information to the log file it creates to track the behavior of each script run. The provenance extraction mechanism processes these log files, filters the entries that contain provenance data, and exports this information to a relational SQL database. Each application execution is launched by a wrapper script that sets up the execution environment. We modified these scripts to also gather runtime information, such as memory consumption and processor load. Additionally, one can define a script that generates annotations in the form of key-value pairs, to be executed immediately before the actual application. These annotations can be exported to the provenance database and associated with the respective application execution. MTCProv processes the data logged by each wrapper to extract both the runtime information and the annotations, storing them in the provenance database. Additional annotations can be generated per script run using *ad-hoc* annotator scripts, as we demonstrate in our case study in section B.4. In addition to retrospective provenance, MTCProv keeps prospective provenance by recording the Swift script source code, the application catalog, and the site catalog used in each script run. Figure B.4 describes the components and information flow of MTCProv.

Provenance information is frequently stored in a relational database. RDBMS's are well known for their reliability, performance and consistency properties. Some shortcomings of the relational data model for managing provenance are mentioned by Zhao et al. [79], such as its use of fixed schemas, and weak support for recursive queries. Despite using a fixed schema, our data model allows for key-value annotations, which gives it some flexibility to store information not explicitly defined in

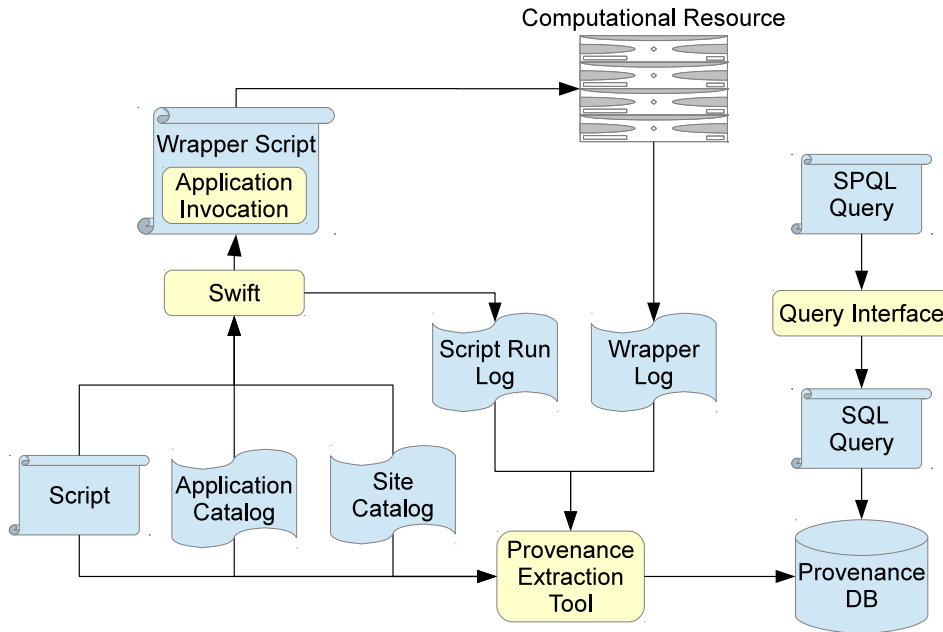


Figure B.4: MTCProv components and flow

the schema. The SQL:1999 standard, which is supported by many relational database management systems, has native constructs for performing recursive queries. Ordonez [173] proposed recursive query optimizations that can enable transitive closure computation in linear time complexity on binary trees, and quadratic time complexity on sparse graphs. Relationship transitive closures, which are required by recursive R^* pattern queries, are well supported by graph-based data models, however as we present in our case study in section B.4, many interesting queries require aggregation of entity attributes. Aggregate queries on a graph can support grouping node or edge attributes that are along a path [174]. However, some provenance queries require the grouping of node attributes that are sparsely spread across a graph. These require potentially costly graph traversals, whereas in the relational data model, well-supported aggregate operations can implement such operations efficiently.

To keep track of file usage across script runs, we record its hash function value as an alternative identifier. This enables traversing the provenance graphs of different script runs by detecting file reuse. Persistent unique identifiers for files could be provided by a data curation system with support for object versioning. However, due to the heterogeneity of parallel and distributed environments, one cannot assume the availability of such systems.

Figure B.5 shows the impact of gathering provenance in Swift. It is based on runs of a sample iterative script using a different number of iterations on each run. The workflow specified in the script has a structure that is commonly found in many Swift applications. In this case, the log file size tends to grow about 55% in size

when provenance gathering is enabled, but once they are exported to the provenance database they are no longer needed by MTCProv. The database size scales linearly with the number of iterations in the script. The initial set up of the database, which contains information such as integrity constraints, indices, and schema, take some space in addition to the data. As the database grows, it tends to consume less space than the log file. For instance, a 10,000 iteration run of the sample script produces a 55MB log file, while the total size of a provenance database containing only this run is 42MB. In addition, successful relational parallel database techniques can partition the provenance database and obtain high performance query processing. The average total time to complete a script run shows a negligible impact when provenance gathering is enabled. In a few cases the script was executed faster with provenance gathering enabled, which indicates that other factors, such as the task execution scheduling heuristics used by Swift and operating system noise, have a higher impact in the total execution time.

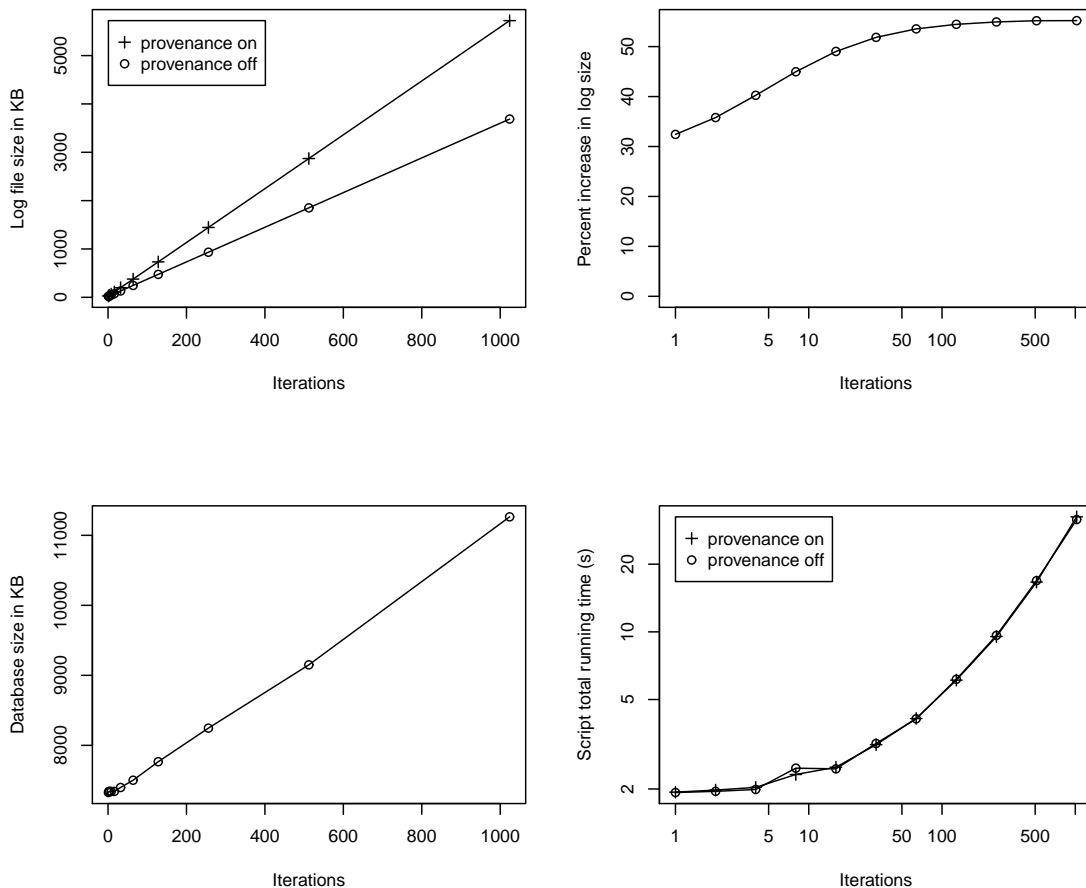


Figure B.5: Impact of gathering provenance.

B.3.2 Query Interface

During the Third Provenance Challenge [64], we observed that expressing provenance queries in SQL is often cumbersome. For example, such queries require extensive use of complex relational joins, for instance, which are beyond the level of complexity that most domain scientists are willing, or have the time, to master and write. Such usability barriers are increasingly being seen as a critical issue in database management systems. Jagadish et al. [175] propose that ease of use should be a requirement as important as functionality and performance. They observe that, even though general-purpose query languages such as SQL and XQuery allow for the design of powerful queries, they require detailed knowledge of the database schema and rather complex programming to express queries in terms of such schemas. Since databases are often normalized, data is spread through different relations requiring even more extensive use of database join operations when designing queries. Some of the approaches used to improve usability are forms-based query interfaces, visual query builders, and schema summarization [176].

In this section, we describe our structured provenance query language, SPQL for short, a component of MTCProv. It was designed to meet the requirements listed in section B.2 and to allow for easier formation of provenance queries for the patterns identified in [92] than can be accomplished with general purpose query languages, such as SQL. SPQL supports exploratory queries [172], where the user seeks information through a sequence of queries that progressively refine previous outputs, instead of having to compose many subqueries into a single complex query, as it is often the case with SQL. Even though our current implementation uses a relational database as the underlying data model for storing provenance information, it should not be dependent on it, we plan to evaluate alternative underlying data models such as graph databases, Datalog, and distributed column-oriented stores. Therefore, in the current implementation, every SPQL query is translated into a SQL query that is processed by the underlying relational database. While the syntax of SPQL is by design similar to SQL, it does not require detailed knowledge of the underlying database schema for designing queries, but rather only of the entities in a simpler, higher-level abstract provenance schema, and their respective attributes.

The basic building block of a SPQL query consists of a selection query with the following format:

```
select (distinct) selectClause
(where           whereClause
(group by       groupByClause
(order by      orderByClause)))
```

This syntax is very similar to a selection query in SQL, with a critical usability benefit: hide the complexity of designing extensive join expressions. One

does not need to provide all tables of the from clause. Instead, only the entity name is given and the translator reconstructs the underlying entity that was broken apart to produce the normalized schema. As in the relational data model, every query or built-in function results in a table, to preserve the power of SQL in querying results of another query. Selection queries can be composed using the usual set operations: union, intersect, and difference. A *select* clause is a list with elements of the form $\langle \text{entity set name} \rangle . \langle \text{attribute name} \rangle$ or $\langle \text{built-in function name} \rangle . \langle \text{return attribute name} \rangle$. If attribute names are omitted, the query returns all the existing attributes of the entity set. SPQL supports the same aggregation, grouping, set operation and ordering constructs provided by SQL.

To simplify the schema that the user needs to understand to design queries, we used database views to define the higher-level schema presentation shown in Figure B.6. This abstract, OPM-compliant provenance schema, is a simplified view of the physical database schema detailed in section B.2. It groups information related to a provenance entity set in a single relation. The annotation entity set shown is the union of the annotation entity sets of the underlying database, presented in Figure B.2. To avoid defining one annotation table per data type, we use dynamic expression evaluation in the SPQL to SQL translator to determine the required type-specific annotation table of the underlying provenance database.

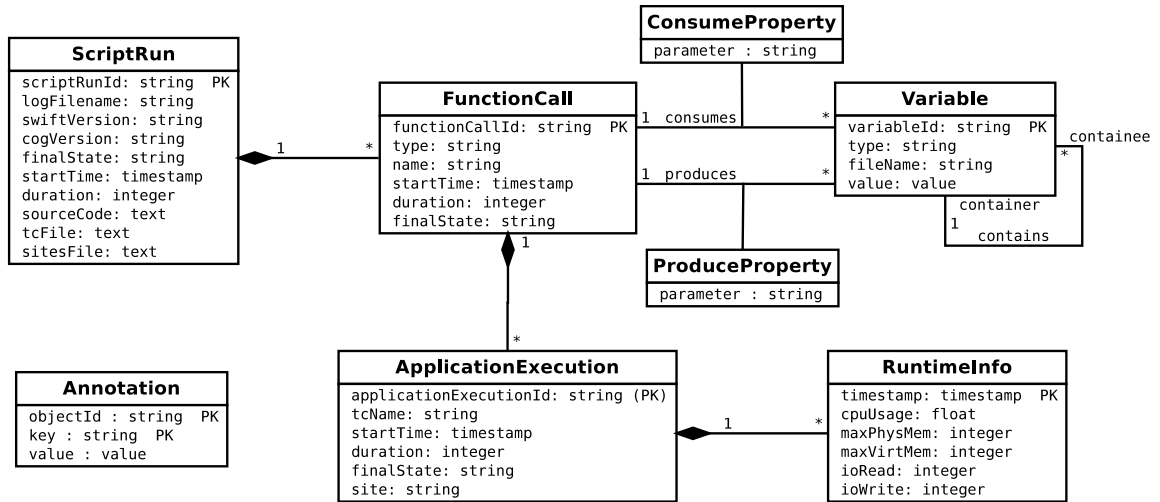


Figure B.6: Higher-level schema that summarizes the underlying provenance database schema.

Most of the query patterns identified in [92] are relatively straightforward to express in a relational query language such as SQL, except for the R* and RCr patterns, which require either recursion or extensive use of relational joins. To abstract queries that match these patterns, we included in SPQL the following built-in functions to make these common provenance queries easier to express:

- `ancestors(object_id)` returns a table with a single column containing the iden-

tifiers of variables and function calls that precede a particular node in a provenance graph stored in the database.

- `data_dependencies(variable_id)`, related to the previous built-in function, returns the identifiers of variables upon which *variable_id* depends.
- `function_call_dependencies(function_call_id)` returns the identifiers of function calls upon which *function_call_id* depends.
- `compare_run(list of <function_parameter=string | annotation_key=string>)` shows how process parameters or annotation values vary across the script runs stored in the database.

The underlying SQL implementation of the `ancestor` built-in function, below, uses recursive Common Query Expressions, which are supported in the SQL:1999 standard. It uses the *prov_graph* database view, which is derived from the *produces* and *consumes* tables, resulting in a table containing the edges of the provenance graph.

```
CREATE FUNCTION ancestors(varchar) RETURNS SETOF varchar AS $$
WITH RECURSIVE anc(ancestor,descendant) AS
(
    SELECT parent AS ancestor, child AS descendant
    FROM   prov_graph
    WHERE  child=$1
    UNION
    SELECT prov_graph.parent AS ancestor,
           anc.descendant AS descendant
    FROM   anc, prov_graph
    WHERE  anc.ancestor=prov_graph.child
)
SELECT ancestor FROM anc $$ ;
```

To further simplify query specification, SPQL uses a generic mechanism for computing the *from* clauses and the join expressions of the *where* clause for the target SQL query. The SPQL to SQL query translator first scans all the entities present in the SPQL query. A shortest path containing all these entities is computed in the graph defined by the schema of the provenance database. All the entities present in this shortest path are listed in the *from* clause of the target SQL query. The join expressions of the *where* clause of the target query are computed using the edges of the shortest path, where each edge derives an expression that equates the attributes involved in the foreign key constraint of the entities that define the edge. While this automated join computation facilitates query design, it does somewhat reduce the expressivity of SPQL, as one is not able to perform other types of joins, such as self-joins, explicitly. However, many such queries can be expressed using subqueries, which are supported by SPQL. While some of the expressive power of SQL is thus lost, we show in the sections that follow that SPQL is able to express, with far

less effort and complexity, most important and useful queries that provenance query patterns require. As a quick taste, this SPQL query returns the identifiers of the script runs that either produced or consumed the file `nr`:

```
select compare_run(parameter='proteinId').run_id where file.name='nr';
```

This SPQL query is translated by MTCProv to the following SQL query:

```
select compare_run1.run_id
from   select run_id, j1.value AS proteinId
       from compare_run_by_param('proteinId') as compare_run1,
       run, proc, ds_use, ds, file
where  compare_run1.run_id=run.id and ds_use.proc_id=proc.id and
       ds_use.ds_id=ds.id and ds.id=file.id and
       run.id=proc.run_id and file.name='nr';
```

Further queries are illustrated by example in the next section. We note here that the SPQL query interface also lets the user submit standard SQL statements to query the database.

B.4 Case Study: Protein structure prediction workflow

We show here how MTCProv is used by scientists to assess and analyze their computational experiments. The workflows were executed on Beagle, a 186-node cluster with 17,856 processing cores; and PADS, a 48-node cluster with 384 processing cores. These machines are located at the Computation Institute, a joint institute of the University of Chicago and Argonne National Laboratory.

Open Protein Simulator (OOPS) [97] is a protein modeling application used for predicting the local structure of loops, and large insertion or chain ends in crystal structures and template-based models. It is used in conjunction with pre-processing and post-processing applications in a high-level workflow that is described in Figure B.1, where the `doLoopRound` workflow activity is a compound procedure. Annotations are gathered by an application-specific script executed after an OOPS run, and stored in the provenance database.

Suppose a scientist runs an application script several times, in what we call a campaign, between April 4, 2010 and August 8, 2010 for a protein modeling competition. After completing the runs, he/she wants to know which ones were recorded in the database for the period. The following query lists runs of the OOPS application script (`psim.loops`) that were executed for the period. This is a simple query for the identifier, start time, and duration attributes of the run entity. The SPQL query is given by:

```
select script_run
where  script_run.start_time between '2010-04-04' and '2010-08-08' and
       script_run.filename='psim.loops.swift';
```

id	start_time	...
psim.loops-20100619-0339-b95u117d	2010-06-19 03:39:15.18-05	...
psim.loops-20100618-0402-qhm9ugg4	2010-06-18 04:02:21.234-05	...
...

One can annotate these runs with the key-value pair (“campaign”, “casp2010”), so that the next queries can be answered with respect to this campaign only. Next, one can list how many times each protein was modeled by each of these runs. This identification of protein modeled is present in the modelIn (see figure B.1) data set. Therefore one way to answer this query is to check the values of the variables that are used as parameter “proteinId”:

```
select  compare_run(function_parameter='proteinId', annotation_key='campaign').proteinId,
        count(*)
where   compare_run.campaign='casp2010'
group by compare_run.proteinId;
```

proteinId	count
T0588end	2
T0601	1
...	...

For the next queries, that illustrate the use of runtime execution information, we display their output graphically in figure B.7. Even though the computational cost of loopModel is highly dependent on the content of the FASTA sequence given as input, the following query comparing the FASTA sequence length with the duration of the loopModel task shows an example of the type of queries enabled by MTCProv:

```
select  compare_run(annotation_key='fasta_sequence_length'),
        avg(function_call.duration)
where   function_call.name='LoopModel'
group by compare_run.fasta_sequence_length;
```

Suppose one wants to analyze why an application execution failed by exploring, for instance, its computational resource consumption. In SPQL this can be obtained with the following queries:

```
select application_execution.id
where  application_execution.final_state='FAILED';

        id
-----
execute:psim-loops-20100618-0402-qhm9ugg:0-1-0-0

select runtime_info
where  application_execution.id='execute:psim-loops-20100618-0402-qhm9ugg:0-1-0-0';
```


Also in figure B.7, one can see a plot, derived from the provenance database, of how many jobs were waiting and how many were executing during a script run. In addition to runtime and domain-specific information, it is possible to design queries that explore lineage information, such as determining all processes and data sets that led to a data set:

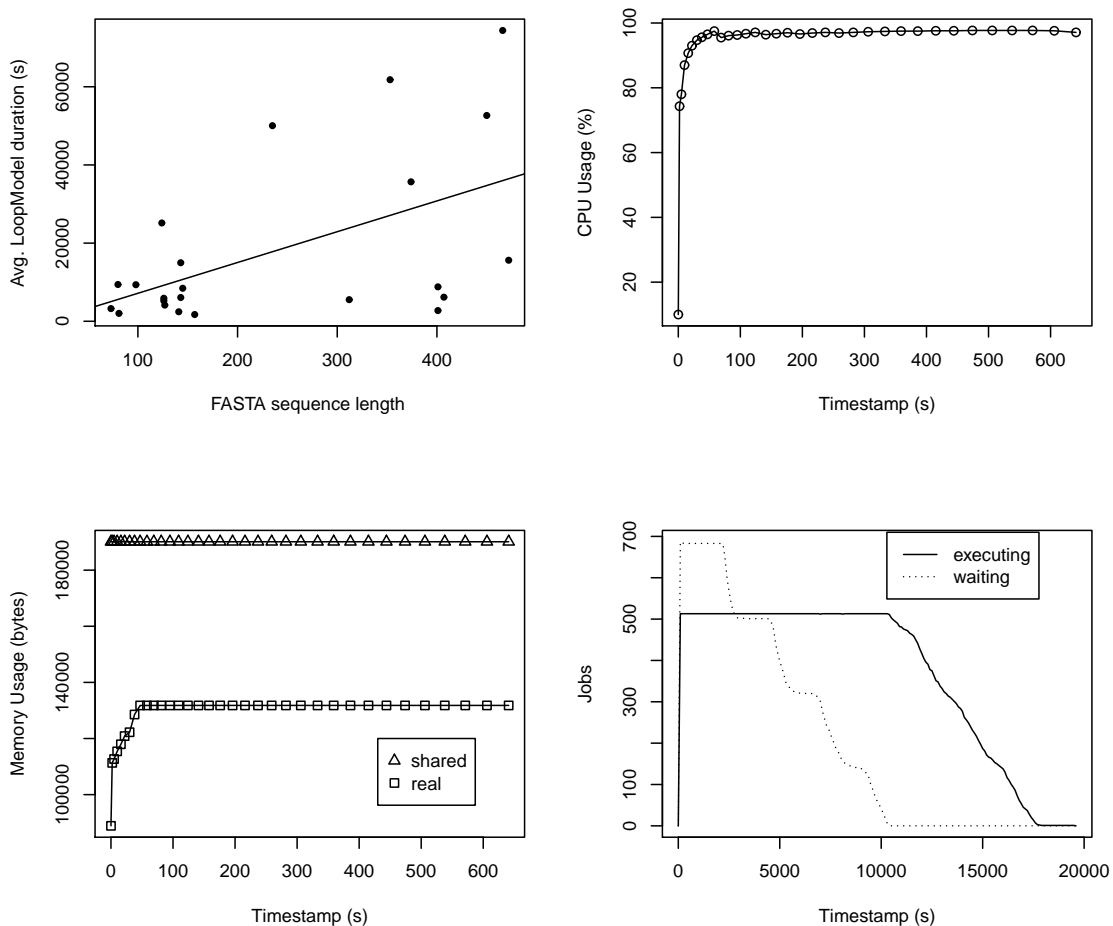


Figure B.7: Plots of query outputs.

```
select ancestors('dataset:20100618-0402-ia0bqb73:72000045');

-----
          ancestor
-----
execute:psim.loops-20100618-0402-qhm9ugg4:451006
dataset:20100618-0402-ia0bqb73:72000039
...
```

These recursive queries can be combined with runtime and domain-specific information to determine, for instance, the range of specific scientific parameters, or the total amount of CPU time used by application executions that preceded a particular dataset in a lineage graph, which can give an estimate of how long would it take to

generate it without parallelism. In SQL, which can use the same built-in provenance functions, one example of such queries can be expressed as:

```
select sum(application_execution.walltime)
from   function_call_dependencies('psim.loops-20100618-0402-qhm9ugg4:451006') as predecessors,
       application_execution
where  predecessors.function_call_id=application_execution.function_call_id;
```

```
          sum
-----
690449.134001970343328
```

In this case, the preceding application executions belong to a same script run. However, this is not always the case, since a file generated by a script run might be used by another one, causing the transitive closure to traverse multiple provenance graphs.

Apêndice C

Security Aspects of Provenance Systems for Distributed Scientific Workflows

C.1 Security Requirements for Provenance Systems

The typical execution of a workflow involves specifying its flow using some mechanism, such as a parallel scripting language or a GUI-based workflow specification tool. Later on, it can be executed by a workflow management system, this involves selecting appropriate computational resources, submitting tasks to these resources, and transferring data. After the experiment is executed, scientists typically face the challenge of analyzing a large number of output data files to understand the outcome of the experiment. Provenance systems are useful in this context since they can help to determine, for instance, which tasks were executed to generate a particular data object, and which parameters were used for these tasks. This provenance data is usually collected and stored during workflow execution, to describe causal relationships between tasks and data (retrospective provenance); or during workflow specification, to describe the planned tasks, and data flow (prospective provenance). In general, provenance data is accessed and analyzed by scientists using a query language, such as SQL. We are enumerated threats to each of these components of a provenance system. Many of these are already taken into account by security frameworks for underlying technologies used by provenance systems, such as databases and grids.

This appendix is based on the papers:

- L. Gadelha, M. Mattoso. Kairos: An Architecture for Securing Authorship and Temporal Information of Provenance Data in Grid-Enabled Workflow Management Systems. *Proceedings of the 4th IEEE International Conference on e-Science (e-Science 2008)*, p. 597-602. IEEE Computer Society, 2008.
- L. Gadelha, M. Mattoso, M. Wilde and I. Foster. Towards a Threat Model for Provenance in e-Science. *Provenance and Annotation of Data and Processes - The Third Provenance and Annotation Workshop (IPAW 2010)*, Troy, USA, p. 277-279. Lecture Notes in Computer Science, vol. 6378, Springer, 2010.

Miles et al. [71] survey requirements for provenance management in e-Science and listed the preservation of integrity and non-repudiation as one of them. Provenance security is a relatively recent research issue [146, 150, 154, 159, 160], found in different areas such as scientific workflows, databases, and storage systems. Provenance data is useful in security audits [161], and there are cases in which the subject of provenance data may lead to privacy concerns [162]. Braun et al. [149] analyze the problem of providing adequate access control techniques to provenance data, observing that it describes causal relationships, that are not adequately protected by commonly used access control techniques. Hasan et al. [150, 154, 159] propose a security solution with the goal of protecting confidentiality and integrity of provenance data. They use asymmetric cryptography for achieving confidentiality and signature-based checksums for achieving integrity. Nagappan et al. [152] present a model for sharing provenance data that uses role-based access control techniques where the user dynamically selects its confidentiality level. A common approach for protecting provenance information [149, 152, 163] is to use access control mechanisms to prevent unauthorized access to this information. None of these approaches allow for non-restricted dissemination of provenance information with maintenance of correct attribution, a requirement in the context of e-Science. Scientists, specially in the life sciences, often avoid sharing details of experiments prior to publishing their results in some academic journal or event, to assure correct attribution of scientific results. During this interval, scientific collaboration is prevented. Therefore, security controls that prevent illegitimate claims of attribution are an important security requirement for provenance systems. These controls must allow the verification of not only who executed an experiment but also when it was executed. A combination of digital signatures and cryptographic timestamps [130–132] were used in the Kairos [73] security architecture for provenance systems to provide these properties. Another desirable security property is fine-grained access control, where scientists can delegate to their collaborators access to provenance data, so it can be read or modified.

In this appendix, we propose Kairos, a security architecture, with the objective of integrating digital signatures and the Time-Stamping Protocol services into grid computing environments in order to protect scientific workflow provenance records.

C.2 Kairos: Protecting Provenance Authorship and Temporal Information

Providing the adequate level of trust to the data generated in experiments is an important aspect of provenance systems, for this purpose it is important to assure the integrity, confidentiality and availability of the information they manage. Current

provenance systems have no support for security. In particular, the protection of data authorship in workflow provenance has been addressed in [146] and [150], however the few proposals available for addressing this issue focus on the use of digital signatures and access control techniques. The objective of this work is to present techniques to secure data authorship and temporal information of provenance records. The main contributions expected are the identification of public key infrastructure techniques that are readily applicable to the protection of this type of information, and the development of a solution, named Kairos, as an architecture that integrates these techniques into SWMS in grid computing environments. The solution aims at scientists that have intellectual property concerns when using distributed workflow management systems.

In this section we provide an overview of the Kairos architecture, discuss implementation issues of this architecture, and describe some experiments performed with the proposed techniques.

C.2.1 Kairos: Secure Provenance Architecture

In this section, we propose Kairos, an architecture with the objective of integrating digital signatures and the Time-Stamping Protocol [130–132] [177] services into provenance systems in order to protect scientific workflow provenance records. These techniques are reviewed in subsection 2.5, they offer a straightforward solution to provide robust security features to protect intellectual property derived from the analysis of provenance data.

C.2.2 Kairos Overview

Following OPM [51], retrospective provenance information consists of assertions about the past, establishing causal relationships between elements of a set artifacts, $\mathcal{D} = \{d_1, \dots, d_l\}$, a set of processes, $\mathcal{P} = \{p_1, \dots, p_m\}$, and a set of agents, $\mathcal{A} = \{a_1, \dots, a_n\}$. Provenance information is given by assertions of the following types:

1. a process used some artifact, or $p_i \xrightarrow{\text{used}} d_j$;
2. an artifact was generated by some process, $d_i \xrightarrow{\text{wasGeneratedBy}} p_j$;
3. a process triggered some other process, or $p_i \xrightarrow{\text{wasTriggeredBy}} p_j$;
4. an artifact was derived from some other artifact, or $d_i \xrightarrow{\text{wasDerivedFrom}} d_j$;
5. a process was controlled by an agent, $p_i \xrightarrow{\text{wasControlledBy}} a_j$.

These assertions may be stored in a variety of ways, such as databases or log files. In scientific workflows, it is important to securely determine the author of these data

artifacts and the date in which they were created in order to validate intellectual property claims. This is a typical application of PKI techniques through the use of digital signatures and cryptographic time-stamping. The use of digital credentials in GSI-based grid environments makes the implementation of the proposed security features for provenance systems relatively straightforward. Currently, these environments use these techniques in client-server authentication, privilege delegation and single sign-on. One would only need to include the missing functionality to perform digital signatures and cryptographic time-stamps. In this section, we propose the Kairos architecture to implement this functionality in grid-enabled provenance systems. Its PKI components, described in figure C.1, are comprised of the following components:

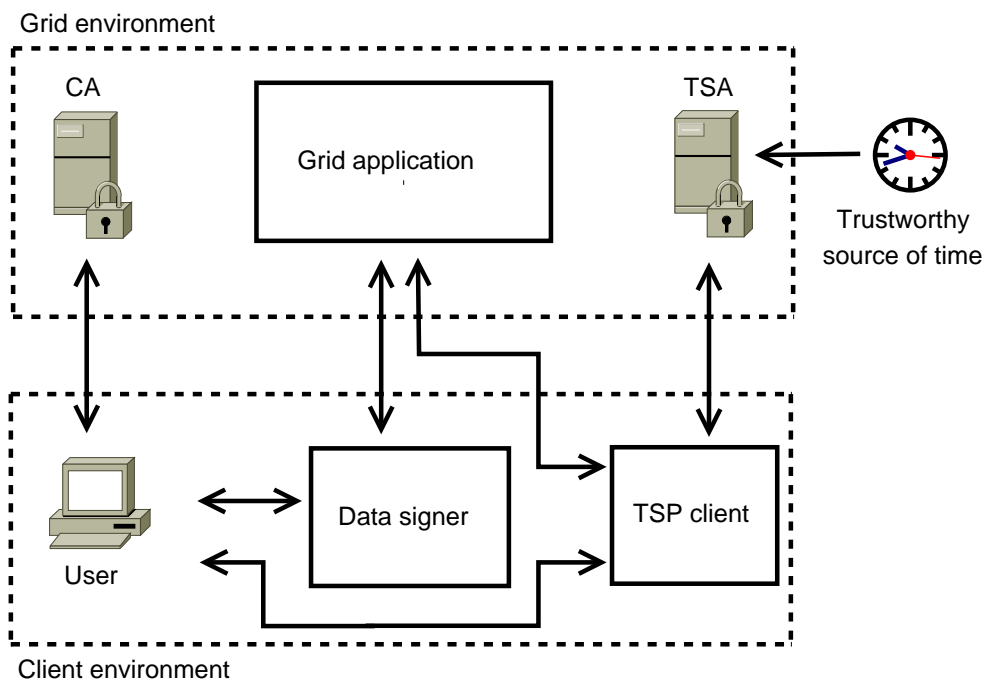


Figure C.1: Kairos: interaction between various PKI components.

- *CA*. It is responsible for issuing digital certificates for the users and the grid resources.
- *TSP client*. It is a client-side application that receives data object hash values from the user. It uses these hash values to build TSP requests that are sent to the TSA.
- *TSA*. It receives TSP requests from the TSP client. It obtains the date and time from a trustworthy source of time, which is appended to the hash value contained in the TSP request. The resulting object is digitally signed with

the TSA's private key. In order to operate, the TSA must have its own digital certificate.

- *Data signer.* It is a client-side application that has access to the user's private key. It receives data objects from the user and digitally signs them using the user's private key. The resulting digital signatures are returned to the user.
- *Grid applications.* Given by computational tasks submitted to the grid environment. It is assumed that they are provenance-aware, as defined in [47]. It is supposed that provenance records contain hash values of the data they refer to, allowing data integrity verification. If one digitally signs the provenance record it would be implicitly digitally signing the data object it refers to.

It can be observed that the TSP client, the TSA and the data signer are effectively the only components added to what would otherwise be a standard grid environment. These components are divided into the *client environment* and the *grid environment*. The client environment is defined as the environment from which the grid is accessed. The grid environment is formed by grid infra-structure and computational resources. As described in figure C.2, the procedure to secure the authorship and the time-stamp of the computational scientific experiment is given by the following steps:

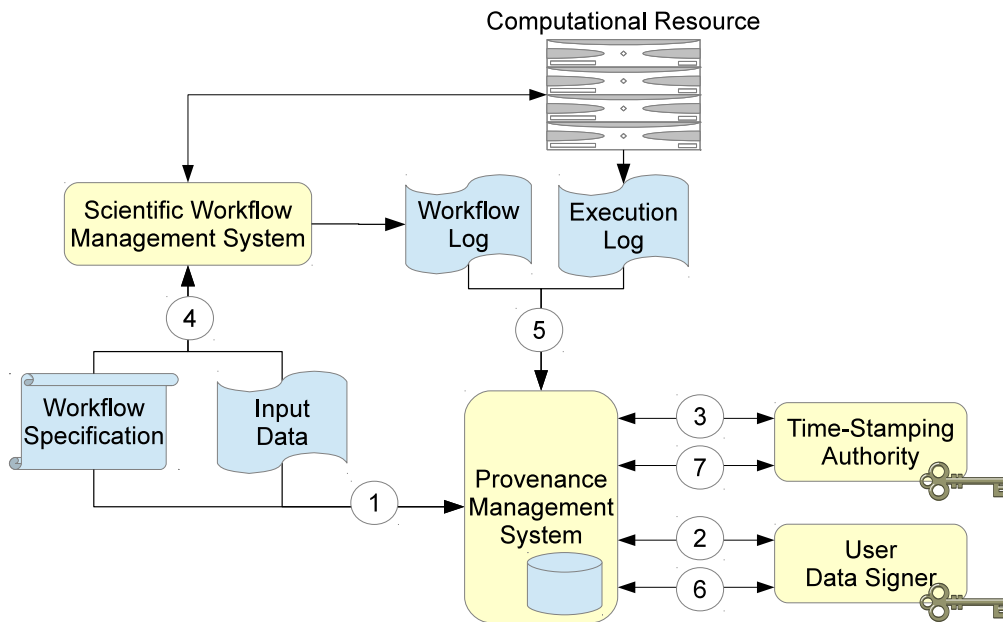


Figure C.2: Kairos: procedure overview.

1. the specification of the scientific workflow and the hash value of its input data are stored in the provenance repository;

2. the specification and input data are digitally signed using the private key of the user;
3. the specification and input data are cryptographically time-stamped using the private key of the TSA;
4. the specification and input data are submitted to the SWMS;
5. after the execution of the workflow, retrospective provenance is stored in the provenance repository;
6. retrospective provenance information is digitally signed using the private key of the user;
7. retrospective provenance information is cryptographically time-stamped using the private key of the TSA;

This procedure effectively protects provenance information regarding data authorship and time-stamp, since any attempt to forge the authorship would invalidate the user's digital signature, and any attempt to forge the time-stamp information would invalidate the TSA's digital signature. An auditor can verify the authorship and time-stamp of some provenance record using the following procedure:

1. The auditor uses the public key from the TSA to verify the appropriate provenance record receipt, i.e. that the hash value h of the user's digital signature of the provenance record existed before the date specified in the time-stamp contained in the receipt.
2. The auditor uses the user's public key to verify the user's digital signature of the provenance record, i.e. that the user is the originator of the provenance record.
3. Finally, the auditor computes the hash value of the data D to compare it with the hash value contained in the provenance record P_D , verifying the integrity of D and that the provenance record is indeed about D .

Kairos offers non-repudiation of information regarding data authorship and time-stamp. It enables applications such as aiding the resolution conflicting intellectual property claims, and the reliable reconstitution of the chronological order of scientific experiments.

Data Type	Signature	Time-Stamp
Log file (text)	CMS [178]	TSP [177]
XML document	\geq XAdES-T [179]	

Table C.1: Applicable security standards.

C.2.3 Kairos Implementation Issues

There are some issues to be considered regarding the implementation of Kairos, such as the representation of the cryptographic time-stamps and digital signatures, the integration with grid environments and provenance recording systems, and the management of the resulting cryptographic data. Some applicable security standards are described in table C.1. Time-stamping and digital signatures techniques are described in a number of different standards, Cryptographic Message Syntax (CMS) digital signatures [178] and the TSP [177] use an ASN.1-based binary format to represent cryptographic data, the first one is widely implemented in web and e-mail clients. In general, ASN.1-based techniques generate detached cryptographic data in binary files, requiring specialized parsing tools for displaying their contents. On the other hand, the use of XML-based security standards, allow the associated cryptographic data to be contained within the document that is being signed or cryptographically time-stamped.

It was observed in [44], that many provenance systems use XML to represent provenance information. Therefore, the use of XML security standards for digital signatures and cryptographic time-stamping seems appropriate in this context. In this subsection we review the XML security standards available for representing digital signatures and cryptographic time-stamps. XML-Signature is a W3C recommendation for representing digital signatures in XML, its syntax and processing rules can be found in [145]. It defines a signature element that can represent signatures over external data (detached signature) or data within the same XML document (enveloping and enveloped signatures). It is closely related to the CMS standard. The XML Advanced Electronic Signatures (XAdES) [179] extends the XML-Signature standard to provide non-repudiation and long-term secure archival of documents. It defines six incremental signature levels, each one builds upon the preceding one providing some additional security feature. For instance, the XAdES-T electronic signature consists of a XML-Signature followed by a cryptographic time-stamp on both the data and the XML-Signature. Kairos, described in section C.2.1, is analog to performing XAdES-T electronic signatures on XML documents containing provenance data.

C.2.4 Demonstration

The tests with the proposed techniques were performed in an environment composed of two machines, one corresponding to the client environment, hosting client-side applications, and the other corresponding to the grid environment, hosting server-side applications. For creating the digital signatures we used OpenSSL [180], which contains an implementation of CMS digital signatures [178]. OpenTSA [181] was also used, it is an implementation of the Time-Stamp Protocol as described in [177], the software is available as a patch to OpenSSL and it is composed of the `ts` application, that is able to build TSP requests and responses; the `tsget` application, which works as a TSP client by sending TSP requests to a TSP server; and the `mod_tsa` module, which allows the Apache web server to work as a TSP server, receiving TSP requests from `tsget`, processing them, and returning TSP responses to `tsget`.

On the server side, a CA was generated with OpenSSL in order to perform the experiments in this section. It was used to issue a digital certificate to the TSA, which is responsible of signing the TSP responses. An Apache web server was configured with `mod_tsa` to work as the TSP server. The server acts as the TSA in figure C.1.

On the client side, it was performed the necessary configuration to access a Globus-based grid, this includes requesting and installing digital certificates for the grid users. A version of OpenSSL was installed with the OpenTSA patch applied. In this case, the OpenTSA applications `ts` and `tsget` act as the TSP client of figure C.1, and the `smime` application of OpenSSL acts as the data signer. The Swift scientific workflow management system [95] was selected for the tests since it is well-integrated with Globus-based grid environments and has functionality to export provenance data from text-based log files to relational databases in order to perform queries. It is capable of exporting OPM graphs [51] in XML format from these databases.

Two scripts, which are executed in the client side, were implemented in order to manage the security of authorship and time-stamp information of the provenance record:

- the `secure_provenance` script is responsible for protecting a particular provenance information. It receives a provenance record `provenance_record.txt` as input and digitally signs it with the user's grid credential, using the `smime` application. The resulting signature, `provenance_record.sig`, is submitted to the `tsget` application, which generates a TSP request and sends it to the TSA. Subsequently, the `secure_provenance` script stores the provenance record receipt (TSP response) in the file `provenance_record.ts`.
- the `verify_provenance` script is responsible for verifying a particular prove-

nance information. It receives a provenance record `provenance_record.txt` as input. It looks for its digital signature, `provenance_record.sig`, which is verified with the user's cryptographic credential, using the `smime` application. Subsequently, it looks for the respective provenance record receipt, `provenance_record.ts`, which is verified using the `ts` application and the TSA's digital certificate.

With the `secure_provenance` script described, it was possible to protect the authorship and time-stamp information of provenance data. Any attempt to forge either the signature or the provenance record results in invalidating them. With the `verify_provenance` script, it was possible to securely verify the validity of the authorship and time-stamp information stated in the digital signature and provenance record receipt files. Some successful tests were performed creating a XAdES-T electronic signature in XML provenance records exported from Swift's provenance database using a XAdES implementation from the Graz University of Technology [182].

The impact of the method depends on the granularity of the digital signatures and time-stamps. If the process is performed once per script run, the impact should be small, since it should add a few seconds to the total running time. If they are performed in a fine-grained mode, such as for each tuple representing a dataset derivation, it can add a considerable amount of time to the total running time.