UNIVERSIDADE FEDERAL FLUMINENSE

FILIPE TADEU SANTIAGO DA SILVA

Uma Abordagem para a Execução de Workflows Científicos Intensivos em Ambientes de Computação em Nuvem Híbridos

Niterói

2019

UNIVERSIDADE FEDERAL FLUMINENSE

FILIPE TADEU SANTIAGO DA SILVA

Uma Abordagem para a Execução de Workflows Científicos Intensivos em Ambientes de Computação em Nuvem Híbridos

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Computação da Universidade Federal Fluminense como requisito parcial para a obtenção do Grau de Mestre em Computação. Área de concentração: Engenharia de Sistemas e Informação

Orientador:
DANIEL CARDOSO MORAES DE OLIVEIRA

Co-orientador:
CRISTINA NADER VASCONCELOS

Niterói

2019

Ficha catalográfica automática - SDC/BEE
Gerada com informações fornecidas pelo autor

S586a Silva, Filipe Tadeu Santiago da
Uma Abordagem para a Execução de WorkflowsCientíficos Intensivos em Ambientes de Computaçãoem Nuvem Híbridos / Filipe Tadeu Santiago da Silva ; DANIEL CARDOSO MORAES DE OLIVEIRA, orientador ; CRISTINA NADER VASCONCELOS, coorientadora. Niterói, 2019.
61 f. : il.

Dissertação (mestrado)-Universidade Federal Fluminense, Niterói, 2019.

DOI: http://dx.doi.org/10.22409/PGC.2019.m.12718954710

1. Workflow científico. 2. Escalonamento. 3. Otimização multiobjetivo. 4. Nuvem de computadores. 5. Produção intelectual. I. OLIVEIRA, DANIEL CARDOSO MORAES DE, orientador. II. VASCONCELOS, CRISTINA NADER, coorientadora. III. Universidade Federal Fluminense. Instituto de Computação. IV. Título.

CDD -

Bibliotecária responsável: Fabiana Menezes Santos da Silva - CRB7/5274

FILIPE TADEU SANTIAGO DA SILVA

Uma Abordagem para a Execução de *Workflows* Científicos Intensivos em Ambientes de Computação em Nuvem Híbridos

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Computação da Universidade Federal Fluminense como requisito parcial para a obtenção do Grau de Mestre em Computação. Área de concentração: Engenharia de Sistemas e Computação

BANCA EXAMINADORA

Prof. D.Sc. Daniel Cardoso Moraes de Oliveira, UFF - Orientador

Profª. D.Sc. Cristina Nader Vasconcelos, UFF - Coorientadora

Prof. D.Sc. Julio Cezar Estrella, ICMC/USP

Prof. D.Sc. Esteban Walter Gonzalez Clua, UFF

Niterói

2019

Aos meus pais.

Agradecimentos

À minha família por sempre ter me lembrado, que apesar das dificuldades nunca devemos desistir dos nossos sonhos.

Aos meus orientadores que apesar do pouco tempo, e muitas tarefas, sempre se mostrou solicito quando eu precisava de uma direção.

Aos meus colegas que me acompanharam durante esta trajetória, me incentivando, dividindo momentos de descontração, estudos e conquistas.

Aos professores da UFF que dividiram seu conhecimento comigo e assim me proporcionaram um crescimento intelectual.

Aos professores Julio Cezar Estrella e Esteban Walter Gonzalez Clua pela presença na banca examinadora.

Resumo

Atualmente a demanda por processamento dos experimentos científicos baseados em simulações computacionais está crescendo exponencialmente, de modo que são necessários esforços por parte dos especialistas em computação para prover suporte para tais experimentos. Muitos avanços foram feitos no decorrer dos últimos anos, os experimentos que antes eram baseados em *scripts* criados pelos próprios cientistas, agora possuem ferramentas que abstraem a parte do código e deixam o cientista se preocupando apenas com a parte conceitual do problema. Tais ferramentas conseguem fazer este experimento ser executado em ambientes de alto desempenho, do inglês *High Performance Computing* (HPC), capturam a proveniência do experimento e ajudam na analise do mesmo. Porém com a evolução da computação, principalmente no uso de *hardwares* aceleradores, os programas utilizados pelos cientistas também evoluíram, para em alguns casos utilizar estas tecnologias para melhorar o seu desempenho. Entretanto, os Sistemas de Workflow que executam esses experimentos ainda carecem de suporte para tais aceleradores. Nosso principal objetivo nesta dissertação é incluir o conceito de *hardwares* aceleradores nesses sistemas utilizados pelos cientistas para execução de experimentos. Para tal precisamos garantir a execução e captura de proveniência desses programas em seus respectivos *hardwares* aceleradores. Além disto, não queremos que o cientista abra mão de uma forma de execução por outra, ou seja, ele informará a atividade, e todas as formas possíveis desta atividade ser executada, e em tempo de execução a ferramenta irá escolher a melhor forma de se executar esta atividade, tirando a responsabilidade desta tarefa do cientista.

Palavras-chave: *Workflow* científico, escalonamento, otimização multiobjetivo, nuvem de computadores.

Abstract

Currently the demand for processing of scientific experiments is growing exponentially, so that efforts are needed by the computer science to provide support for such experiments. Many advances have been made over the past few years, experiments that were previously based on scripts created by scientists themselves now have tools that abstract the part of the code and leave the scientist concerned with the conceptual part of the problem only. These tools can make this experiment run in High Performance Computing (HPC) environments, capture the provenance of the experiment and help analyze it. But with the evolution of computing, especially in the use of accelerators, the programs used by scientists have also evolved, to use these technologies to improve their performance. Our main goal in this dissertation is to include the concept of accelerators in these tools used by scientists to perform experiments. For this we need to guarantee the execution and provenance‘s capture of these programs in their respective hardware. In addition, we do not want the scientist to give up one form of execution for another, **i.e.**, it will inform the activity, and all possible forms of this activity will be executed, and at run time the tool will choose the best way to perform this activity, taking the responsibility of this task.

Keywords: Scientific workflow, scheduling, multi-objective optimization, cloud computing.

Lista de Figuras

2.1 Ciclo de vida do experimento Cientifico. [34] 8

2.2 O Modelo PROV-Wf. 10

2.3 Ciclo de vida do experimento Cientifico. 12

2.4 Arquitetura *Body* e *Head* do SciCumulus. 15

2.5 Exemplo de Workflow. 15

2.6 Exemplo de execução com o DYN-FAF. 16

2.7 Exemplo de execução com o DYN-FTF. 17

3.1 Ampliação do PROVWf. 22

3.2 Proveniência de Múltiplos Aceleradores. 23

3.3 Exemplo de Modelagem Híbrida 26

3.4 Exemplo Escalonamento FAF 28

3.5 Exemplo Escalonamento FTF 29

4.1 Sciphy - Modelo de Execução 35

4.2 Sciphy - Modelagem 36

4.3 Sciphy - Modelagem (CPU & GPU) 36

4.4 Tempo de Execução MSA 38

4.5 Tempo de Execução Total 38

Lista de Tabelas

4.1 Experimentos Realizados 36

4.2 Experimentos Realizados 37

4.3 Experimentos Realizados 37

4.4 Experimentos - MSA (CPU/CPU e GPU) 39

Lista de Abreviaturas e Siglas

Sumário

1 Introdução **1**

1.1 Questão de Pesquisa 3

1.2 Contribuições 4

1.3 Organização da Dissertação 4

2 Fundamentação Teórica **5**

2.1 Experimento Científico 5

2.1.1 Ciclo de Vida de um Experimento Cientifico 7

2.2 Proveniência de Dados 8

2.2.1 PROV-Wf 10

2.3 O SciCumulus 11

2.3.1 Representação de *Workflows* Científicos 13

2.3.2 Modelo de Execução 14

2.4 GPGPU 17

3 Abordagem Proposta **19**

3.1 Arquitetura Proposta 19

3.1.1 Representação de *Workflows* Científicos 20

3.2 Modelo de Proveniência 21

3.2.1 Ampliação do modelo de proveniência 22

3.2.2 Extração e Armazenamento de Dados de Aceleradores 23

3.3 Execução do *Workflow* 24

3.3.1 Inicialização 24
3.3.2 Escalonador de Tarefas 25
3.3.3 Captura de Proveniência 29
3.4 Modelo de Custo 30
3.4.1 Tempo de Execução 30
3.4.2 Custo Financeiro 32

4 Avaliação Experimental 34
4.1 Sciphy 34
4.2 Experimentos 35
4.2.1 Ambiente 35
4.2.2 Execução 36
4.2.3 Resultados 37

5 Trabalhos Relacionados 40

6 Conclusão e Trabalhos Futuros 43
6.1 Suporte a novos aceleradores 43
6.2 Escalonamento das Tarefas 44

Referências 45

Capítulo 1

Introdução

O processo de experimentação é uma das formas usadas para apoiar as teorias baseadas em um método científico [44]. De forma a considerar um corpo de conhecimento como sendo de fato "científico", sua veracidade e validade devem ser comprovadas [53] [34]. No método científico, um experimento científico consiste na montagem de um protocolo concreto a partir do qual se organizam diversas ações observáveis, direta ou indiretamente, de forma a corroborar ou refutar uma dada hipótese científica que foca em estabelecer relações de causa/efeito entre fenômenos [5] [30].

A evolução da ciência da computação nas últimas décadas permitiu a exploração de novos tipos de experimentos científicos baseados em simulação [19], os chamados experimentos *in silico*. De fato, a computação vem sendo um grande acelerador para a ciência e tem gerado uma sobrecarga de informação em diversos domínios de pesquisa [14] como a biologia, química e física.

Um experimento *in silico* é caracterizado pelo uso de simulações do mundo real, realizadas em ambientes virtuais e que são baseadas em modelos complexos. Em grande parte dos casos, estes modelos se referem ao encadeamento de programas que são utilizados durante as simulações. Cada execução de programa pode consumir e produzir um grande volume de dados. A execução de experimentos baseados em simulação é apoiada por diversas técnicas e abordagens, como os workflows científicos [19] [51], que é uma abstração que permite a composição de programas em uma sequência de execução com o objetivo de gerar um resultado final.

Era comum um cientista fazer um *script* para modelar e executar seu experimento, de forma *ad-hoc* [14], o que tornava o experimento muito mais lento e caro, uma vez que o cientista neste cenário necessita ter um conhecimento computacional considerável,

e o reuso de código não é algo trivial, visto que o código foi feito se pensando em uma determinada pesquisa, e por este motivo pode não ser reutilizado em outras pesquisas. Além disto o cientista ficava com a obrigação de modelar a captura de dados para responder perguntas como: Quem criou esse dado? Que consumiu este dado? Quando ele foi alterado e por quem? Este processo não só consumia tempo, mas também era muito suscetível a erros [14]. Atualmente, os workflows científicos são especificados, executados e monitorados por sistemas complexos chamados de Sistemas de Gerência de Workflow Científicos (SGWfC) [55] [40] [33] [39] que possuem um motor de execução acoplado, responsável por invocar cada um dos programas do fluxo sem a necessidade do cientista se preocupar com a gerência do workflow.

Existem diversos desafios inerentes a execução de workflows pelos SGWfCs. Um desafio é a execução do workflow como um todo, nesse sentido a pesquisa é feita sobre como otimizar a execução do *workflow* científico, outra é mais focada nos programas que compõem o *workflow* científico, ou seja, como fazer um programa ser executado de forma mais rápida possível, para não criar gargalos na execução do *workflow*.

Como muitos desses workflows são de larga escala, para processar, analisar e entender estes dados em tempo hábil, frequentemente é necessário reunir ferramentas computacionais complexas, como *frameworks* e infraestruturas especializados, *clusters* de computadores, grades computacionais, nuvens computacionais e serviços *web*.

Enretanto, com o advento e aumento da popularidade das unidades gráficas, do inglês *Graphics Processing Unit* (GPU), e principalmente com o conceito de unidade de processamento gráfico de propósito geral, do inglês *General Purpose Graphics Processing Unit* (GPGPU), diversos provedores de nuvem como a Amazon AWS tem disponibilizado Máquinas virtuais com GPGPUS a um determinado custo, o que chamamos de Nuvem Híbrida.

Além disso, diversas aplicações científicas tem disponibilizado versões em CUDA e OPENCL - linguagens de programação em GPGPU, como por exemplo o GPU-BLAST [45] e o CUDA CLUSTALW [28]. Tais aplicações podem ser usadas em *workflows* para acelerar a execução desses experimentos. Dessa forma, os cientistas identificaram que parte das aplicações usadas em seus *workflows* poderiam se adaptados para esta linguagem [32] [10], e assim tomar proveito desta unidade aceleradora para otimizar seus experimentos.

É importante ressaltar que no caso dos *clusters*, grades e nuvens, devido ao nível de maturidade alcançado pelas tecnologias, existem dezenas de propostas de abordagens para oferecer apoio à execução e a gerência de *workflows* nestes ambientes [2] [3] [4] [11]

[21] [38] [17] [41] [48] [49] [52] [58]. Entretanto o mesmo não pode ser dito em relação às nuvens de computadores híbridas, que consideram tanto vCPUS e GPGPUS em suas máquinas virtuais.

O SGWfC deve ser capaz de receber uma especificação do *workflow* contendo aplicações que rodam em CPU e aplicações que executam em GPGPUS e executar o mesmo de forma eficiente, além de coletar os dados de proveniência associados. Como a complexidade dos *workflows* aumenta exponencialmente (por exemplo, a exploração de milhares de parâmetros e, em muitos casos usando estruturas de repetição ao longo de dezenas de atividades complexas), executar esses *workflows* exige a aplicação de técnicas de paralelismo. Existem vários *workflows* científicos que foram projetados para serem executadas em paralelo (explorando varredura de parâmetros), que demandam recursos de computação de alto desempenho, como o Montage, um *workflow* para análise de dados astronômicos [29], *workflows* de análise de cristalografia de raios-X [15], *workflows* para detecção de genes ortólogos [12] e *workflows* para análise filogenética [37], sendo este último o estudo de caso desta dissertação.

Entretanto, executar estes *workflows* em nuvens híbridas ainda é um problema em aberto, porém importante de ser resolvido. Essas execuções trazem a tona problemas que devem ser tratados para, de fato, alcançarmos a gerência da execução de um *workflow* científico em um ambiente de nuvem híbrida de forma satisfatória. Um problema importante é a questão do escalonamento de atividades, que é o foco dessa dissertação.

1.1 Questão de Pesquisa

Desta forma, a questão de pesquisa a ser investigada nesta dissertação é:

"No contexto de experimentos científicos em larga escala executados em ambientes de nuvens computacionais híbridas com CPUS e GPGPUS, é possível promover a gerência distribuída do workflow científico por meio da adoção de uma abordagem que controle a execução de atividades em paralelo na nuvem e utilize um modelo de proveniência que represente descritores tanto do workflow quanto do ambiente em que o mesmo está sendo executado, tornando possível o aumento de desempenho da execução e a análise de proveniência destes experimentos por parte dos cientistas?".

1.2 Contribuições

Para responder essa questão de pesquisa, nesta dissertação estendemos o SGWfC SciCumulus, que tem foco na execução de workflows em nuvens de computadores, para trabalhar tanto com CPUS quanto GPUS nos workflows. Desta forma, foi projetado e desenvolvido o SciCumulus-GPU, um arcabouço para a execução paralela de workflows científicos em nuvens computacionais híbridas. Esta dissertação contextualiza sua pesquisa na área de "ciência apoiada pela computação" (do inglês e-science ou cyberinfrastructure), contribuindo nesta área por apresentar:

- Um modelo de custo ponderado para escalonamento de tarefas de workflows científicos em nuvens híbridas;
- Uma abordagem de escalonamento gulosa e adaptativa para explorar o espaço de escalonamentos alternativos considerando o uso do modelo de custos proposto;
- Uma extensão do modelo de proveniência PROV-Wf [18] que considera os dados descritivos do ambiente de nuvem híbrida e os dados referentes à estrutura e execução dos workflows;
- Uma avaliação experimental do workflow de análise filogenética com o SciCumulus-GPU. Para esta avaliação experimental executamos uma versão modificada do SciPhy [37] que contém aplicações que executam em GPUs, em um cluster virtual híbrido na nuvem Amazon AWS, mostrando os benefícios da abordagem proposta.

1.3 Organização da Dissertação

Além deste capítulo introdutório, esta dissertação está organizada em outros 5 capítulos. O capítulo 2 apresenta conceitos relacionados a experimentos científicos, *workflows* científicos, proveniência de dados e unidades de processamento gráfico. O capítulo 3, referente a abordagem proposta, detalha o fluxo de execução e de captura de proveniência para experimentos realizados em nuvens híbridas. O capítulo 4 apresenta a avaliação experimental da abordagem proposta, avaliando as mudanças realizadas no algoritmo de escalonamento pensando nos *hardwares* aceleradores. O capítulo 5 apresenta os trabalhos relacionados. Finalmente o capítulo 6 conclui a dissertação apresentando os principais resultados alcançados e os desdobramentos para diversos trabalhos futuros.

Capítulo 2

Fundamentação Teórica

2.1 Experimento Científico

O processo de experimentação é uma das formas usadas para apoiar as hipóteses baseadas em um método cientifico. Neste cenário as duas variáveis mais importantes para são os dados utilizados e os conceitos envolvidos no desenvolvimento da hipótese.

A principio o cientista considera a sua hipótese como verdadeira, e analisa as consequências lógicas de tal hipótese, feito isto o cientista cria experimentos que possam validar sua hipótese [44], tais experimentos podem comprovar ou não sua hipótese, e, dependendo deste resultado a hipótese pode ser anulada, alterada ou confirmada.

Por fim para tal hipótese ser aceita pela comunidade científica ela deve ser passível de reprodução [44], ou seja, outros cientistas devem ser capazes de validar a hipótese fazendo experimentos semelhantes e chegando a resultados semelhantes que a comprovem.

Existem diversos tipos de experimentos científicos [54], que se diferem em base pelo ambiente que tal experimento é realizado, sendo classificados em :

- ***In vivo*** São experimentos feitos em ambientes naturais, sobre agentes vivos, sem nenhuma simulação envolvida.
- ***In vitro*** São os experimentos feitos em ambientes controlados que tentam simular um ambiente natural, a qualidade do experimento é também dependente da qualidade de tal simulação.
- ***In virtuo*** São experimentos feitos em ambientes simulados, com condições simuladas, o cientista pode alterar tal experimento durante a sua execução, conforme o ambiente reagir as mudanças realizadas.

- ***In silico*** Tanto a pesquisa quanto o ambiente da mesma são modelados computacionalmente e o cientista não tem acesso ao processo, somente aos dados gerados no decorrer do experimento.

Nosso foco nesta dissertação são os experimentos *in silico*. Esses experimentos cada vez mais requerem um processamento de alto desempenho para serem executados em tempo viável [25], devido ao grande volume de dados que possuem.

Os experimentos *in silico* em sua grande maioria são representados pelo encadeamento de múltiplas combinações de programas [53] , onde a saída de um programa é normalmente consumida como entrada para o programa sucessor no encadeamento.

Esses programas podem consumir grande quantidade de dados, e são executados em ambientes de alto desempenho. Em tais experimentos cada programa pode ser executado consumindo um grupo específico de parâmetros e dados, cada qual com sua própria semântica e sintaxe.

Para auxiliar na execução de tais experimentos foi criado o conceito de *workflow* científico. Tal conceito foi originado na área de automação de escritórios [24] por volta de 1970 com foco em organizar a geração, armazenamento e compartilhamento de documentos dentro de uma instituição.

Nas últimas décadas esse conceito foi utilizado por diversas áreas da ciência para abstrair e modularizar os experimentos, que outrora eram executados por *scripts* criados pelos cientistas. Esta prática gerava uma grande quantidade de erros [14], tanto na hora da execução do projeto, quanto na hora da captura dos dados gerados por tais experimentos.

O termo *workflow* científico é utilizado para descrever *workflows* em algumas áreas da ciência, nas quais se compartilham características de manipulação de grande volume de dados [25] [14], heterogeneidade na representação dos dados, e demanda por alto poder de processamento.

Esses *workflows* são montados, controlados e executados por Sistemas de Gerência de *Workflows* Científicos (SGWfC). Existem diversos SGWfC disponíveis, como o WASA [55], Taverna [39], Kepler [33], SciCumulus [40], Pegasus [20], Swift/T [59], VisTrails [8], dentre outros, e cabe ao cientista analisar os prós e contras de cada um deles para escolher o que melhor convém a sua pesquisa.

O conceito de experimento científico engloba o conceito de *workflow*, de modo que eles não podem ser tomados como um sinônimo, um experimento científico pode ser composto

de vários *workflows*, e utilizar diversos SGWfC. E mesmo um experimento que possua um único *workflow*, eles ainda não são sinônimos, um *workflow* é um ensaio (do inglês *trial*) realizado no contexto de um experimento cientifico para avaliar uma ação controlada. De modo que um experimento científico é definido pelo conjunto de ensaios realizados, ou seja, pelas diversas execuções do *workflow*.

Por simplificação, nesta dissertação um *workflow* científico é definido como um grafo acíclico dirigido (do inglês *Directed Acyclic Graph* ou DAG) chamado W(A, Dep). Os nós (A = a_1, a_2, ...,a_n) em W correspondem a todas as atividades do *workflow* a ser executado e as arestas (Dep) estão associadas à dependência de dados entre as atividades de A.

Desta forma, dado a_i | (1 $\leq$ i $\leq$ n), consideremos como I = i_1, i_2, ...,i_m o conjunto possível de dados de entrada para a atividade a_i, então $Input(a_i) \supset$ I. Além disso, consideremos O como sendo o conjunto possível de dados de saída produzidos por a_i, então $Output(a_i) \supset$ O. Uma atividade específica a_i é modelada como um $a_i(time)$ onde *time* é o tempo total de execução de a_i. A dependência entre duas atividades é modelada como dep(a_i, a_j, ds) $\leftrightarrow \exists\, O_k \in Input(a_j) \mid O_k \in Output(a_i)$ e ds é o volume de dados transferidos entre essas duas atividades (independentemente da unidade de medida utilizada).

2.1.1 Ciclo de Vida de um Experimento Cientifico

Na Figura 2.1 as principais fases do ciclo de vida de um experimento científico podem ser identificadas, a execução, a composição e a análise [34]. Cada fase possui um subciclo independente que é executado em momentos diferentes do curso do experimento.

A fase de composição é responsável pela criação do *workflow*, nesta fase o cientista escolhe quais programas serão executados, e a sequência que eles deverão ser executados, além de definir qual tipo de entrada eles precisam e quais saídas eles irão gerar. Ainda podemos dividir tal fase em duas subfases, sendo elas concepção e reuso, na primeira o cientista irá criar o *workflow*, já na segunda o cientista irá adaptar um *workflow* já existente para atender sua necessidade.

A fase de execução é responsável por concretizar a fase de concepção, ou seja, alimentar o *workflow* com os dados passados, e o executar de acordo com o que o cientista planejou na etapa anterior, esta fase possui também duas subfases, distribuição e monitoramento. A primeira responsável pela execução do *workflow* em ambientes de Programação de Alto Desempenho (PAD), devido as necessidades de desempenho. A subfase de monitoramento está ligada ao cientista está a par da execução do experimento, uma vez que ele pode

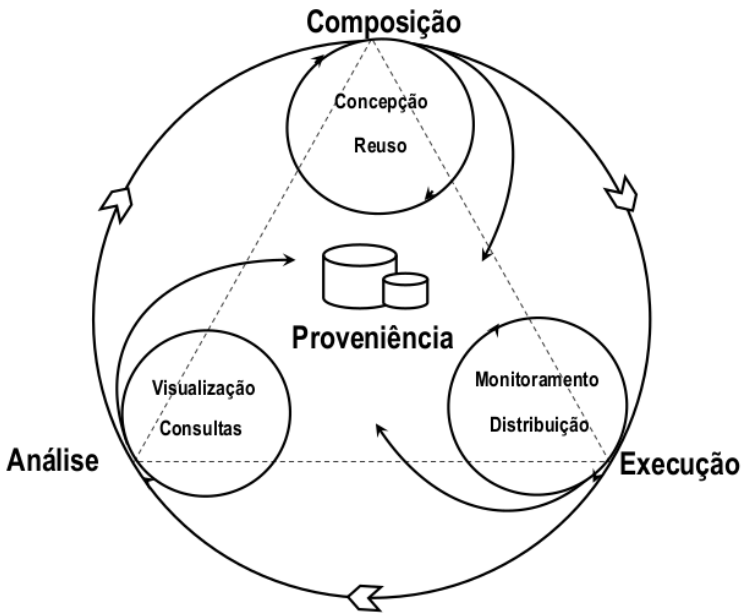

Figura 2.1: Ciclo de vida do experimento Cientifico. [34]

demorar muito tempo para ser executado.

Por fim temos a fase de análise, que ocorre após a execução do *workflow*, e é responsável por analisar os dados gerados pelo mesmo. Em tal fase o cientista irá analisar os resultados do seu experimento através dos dados de proveniência que foram capturados durante a execução do *workflow*.

Tal fase irá determinar se a hipótese foi aceita ou não, caso não tenha sido aceita o cientista poderá reformular a hipótese, o *workflow* e o reexecutar ou simplesmente abandonar a hipótese inicial, caso tenha sido aceita o cientista provavelmente precisará executar o *workflow* mais algumas vezes com diferentes dados para que assim confirme sua hipótese, e todas as execuções deverão estar atreladas ao mesmo experimento.

2.2 Proveniência de Dados

O termo proveniência, segundo o dicionário Michaellis [1], é definido como lugar de onde uma coisa provém, emana ou se deriva; fonte, origem, precedência.

No contexto de um experimento científico a proveniência engloba todos os dados obtidos com os experimentos, porém não apenas isto. São dados que simbolizam um experimento [47], ou seja, estão ligados ao experimento e devem ser atrelados de alguma forma ao mesmo.

A definição dada para proveniência no contexto de experimentos científicos dada por Buneman [7] diz que a proveniência é a descrição da origem de um dado e o processo pelo qual este chegou a um banco de dados. Goble [27] resumiu as principais funcionalidades da proveniência em quatro pontos, sendo eles:

- **Qualidade dos Dados** Baseando-se na origem dos dados e suas diversas modificações no decorrer da execução do *workflow* podemos estimar não só a qualidade dos dados, mas também sua confiabilidade
- **Auditoria dos Caminhos** Com a proveniência é possível traçar as rotas tomadas pelos dados, verificar se houveram falhas na geração de tais dados e a utilização de recursos para os gerar.
- **Verificação de Atribuição** Mantém informações sobre o time que criou o experimento e é responsável por tais dados, para assim facilitar o re-uso e a devida citação dos criadores.
- **Informacional** Permite realizar diversas consultas baseadas nos descritores de origem, para assim ser possível descobrir os dados gerados e os contextualizar.

Exitem dois tipos de proveniência a prospectiva e a retrospectiva [23] [13] [14]. A proveniência prospectiva é responsável por armazenar o passo-a-passo do experimento que são necessários para gerar uma informação ou uma classe de informações. Já a retrospectiva captura os passos que foram dados assim como informações do ambiente em que o experimento foi executado.

Em outras palavras a proveniência prospectiva diz respeito ao plano de execução do *workflow*, enquanto a retrospectiva nos informa a respeito de suas execuções.

O modo de captura e representação da proveniência de um experimento vem sendo debatido a muito tempo no meio científico devido a sua importância para a analise do experimento, do teste de reprodutibilidade, e do reuso do mesmo.

De modo que em 2006 um grupo de pesquisadores proeminentes na área organizou um *workshop* (o IPAW [6]) para que a comunidade pudesse discutir como seria a melhor maneira de representar a proveniência gerada por um *workflow* científico.

Após isto dois desafios foram propostos, o primeiro [36] para criar modelos de proveniência para um *workflow* de ressonância magnética, e o segundo [35] para que fosse

estabelecido um sistema de interoperabilidade entre os *workflows* criados no primeiro desafio.

Ao final dos desafios os times chegaram a um consenso das principais representações de proveniência, dando origem ao modelo de proveniência aberto, OPM [35] (do inglês *Open Provenance Model*).

Porém, com a chegada do PROV [56] o OPM se tornou obsoleto, principalmente por sua representação ser baseada em grafos, enquanto o PROV utiliza o modelo relacional. Além disto o PROV possui ontologias associadas a sua representação, o que facilita a interoperabilidade. Tanto o OPM quanto o PROV expressam suas relações causais entre Processos, Agentes, Artefatos e Papéis.

2.2.1 PROV-Wf

Uma das extensões do PROV é o PROV-Wf, que tem como foco principal trazer soluções para ambientes de *workflows*, trazendo propostas como proveniência em tempo de execução, auxiliar a lidar com erros nas tarefas, monitorar o status de execução do *workflow* e escalonar as tarefas do *workflow* de modo que melhor atenda as necessidades do cientista.

O PROV-Wf é utilizado para representar tanto a proveniência prospectiva quanto a retrospectiva, ou seja, ele fornece os detalhes das execuções das diversas execuções de um *workflow* científico. Ele é composto por três partes principais, a estrutura do experimento, a execução do experimento, e a configuração do ambiente de execução.

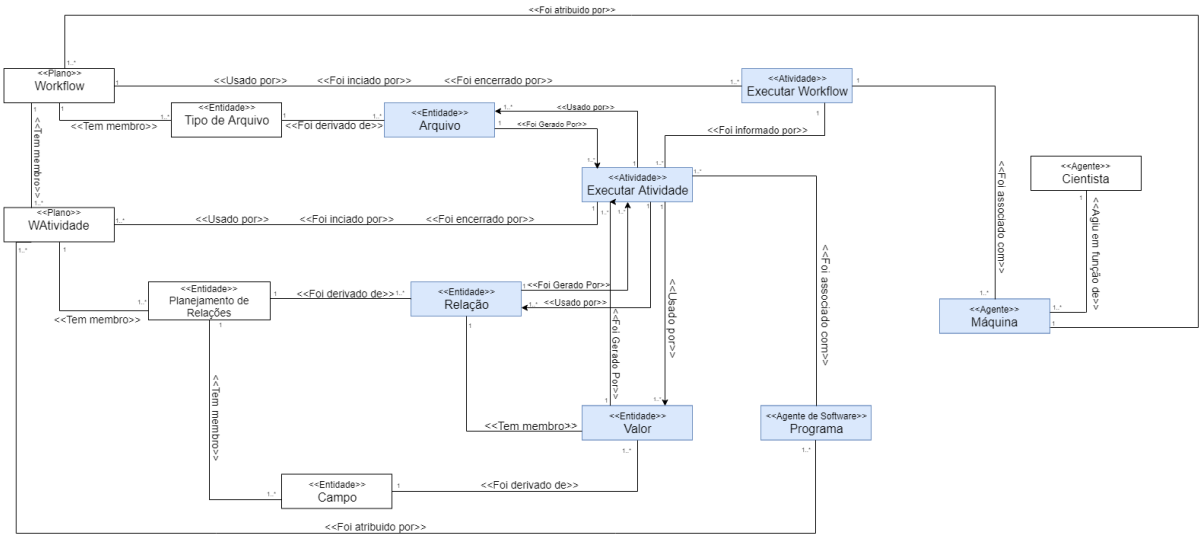

Figura 2.2: O Modelo PROV-Wf.

O agente Cientista representa uma pessoa que irá executar o *workflow*, este também está associado ao agente máquina. O agente máquina estabelece uma associação com um *workflow* (*i.e.* Plano do *Workflow*), que por sua vez é composto por um conjunto de atividades (*i.e.* Plano WAtividades). Cada atividade é responsável por executar um programa em uma máquina com as configurações especificadas.

A invocação de um programa em um *workflow* (*i.e.* Instância da Execução) pode ser setada como um conjunto de valores para serem consumidos. Para expressar todos os dados que são consumidos e produzidos pela execução de uma instância a entidade Esquema de Relacionamento está associado com um esquema e pode ser definido por múltiplos campos. Cada campo (*i.e.* Entidade Campo) descreve o significado de cada parâmetro associado a um programa que está relacionada a uma WAtividade. A entidade Valor dafine o conjunto de valores de um campo, cada qual associada a uma instância de execução. A entidade Arquivo representa todos os arquivos consumidos e produzidos pela execução de um *workflow*, a entidade Tipo de Arquivo representa os tipos de arquivos que o *workflow* espera consumir e produzir.

As entidades com associações com consumo e criação de arquivos e valores possuem duas associações direcionais PROV, sendo elas respectivamente, usado por (consumo), e gerado por (produção).

2.3 O SciCumulus

O SciCumulus [40] é um SGWfC voltado para execuções de experimentos que manipulam uma grande quantidade de dados e foi especialmente desenvolvido para ambientes de nuvem. A motivação do seu desenvolvimento foi o bom desempenho do mediador Hydra e pelo motor de execução Chiron na distribuição de atividades de varredura de parâmetros de *workflows* científicos em *clusters* de computadores.

Com o objetivo de isolar os cientistas da tarefa de distribuir as atividades em ambientes de nuvem, o SciCumulus oferece mecanismos para oferecer paralelismos por varredura de parâmetros. Tal tarefa quando feita de forma *ad-hoc* é demasiadamente trabalhosa devido ao número de *cloud activities* que precisam ser gerenciadas. Para prover a infraestrutura necessária para realizar tal tarefa o SciCumulus possui uma arquitetura de quatro camadas, sendo elas:

- *Camada Cliente* Esta camada tem a responsabilidade de ser a interface entre o

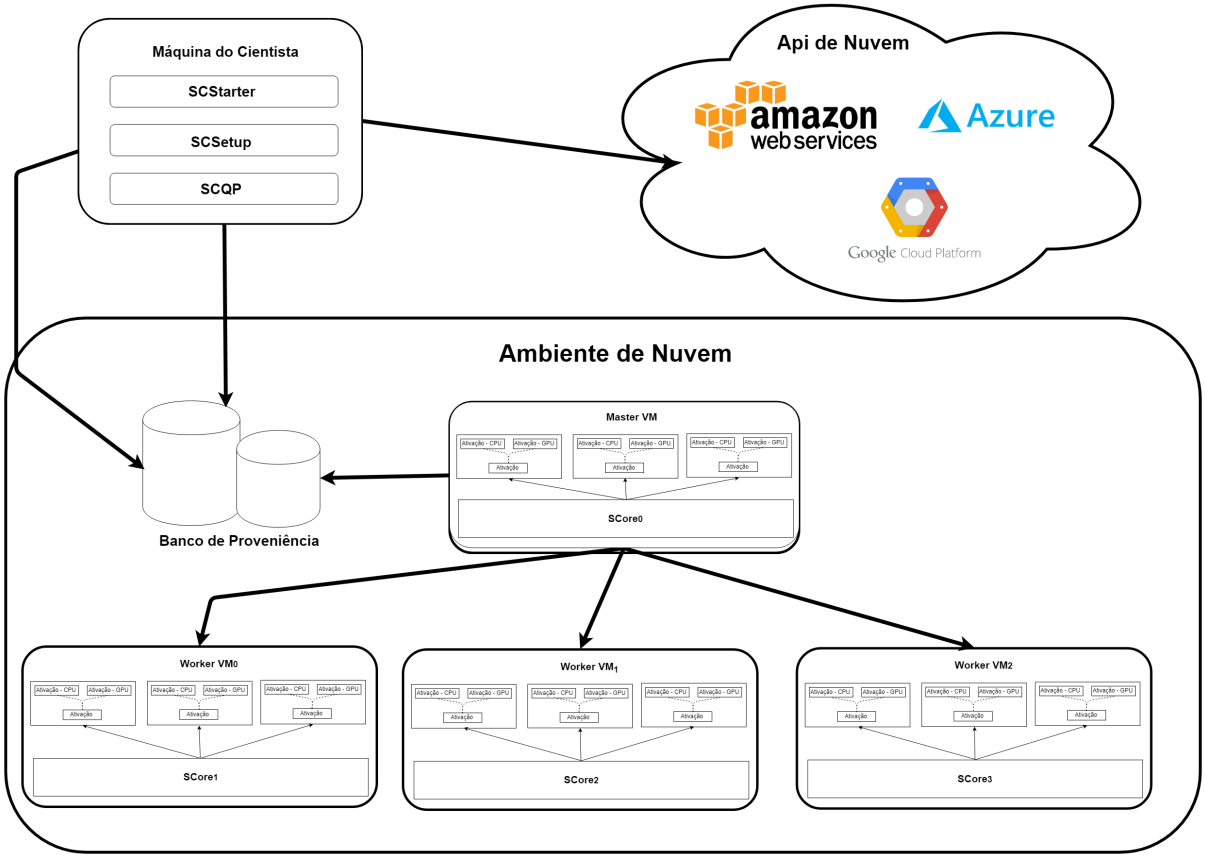

Figura 2.3: Ciclo de vida do experimento Cientifico.

ambiente de nuvem e o SGWfC. Seus componentes são instalados na máquina do cientista e despacham as atividades do *workflow* para serem executadas em um ambiente de nuvem.

- *Camada de Distribuição* Tem a principal tarefa de gerenciar a distribuição das *cloud activities* em uma ou mais máquinas instanciadas preferencialmente em um ambiente de nuvem.

- *Camada de Execução* É responsável pela execução do programa de uma *cloud activity* e pela captura da proveniência gerado por este programa. Seus componentes são instalados nas várias máquinas virtuais envolvidas na execução do *workflow*.

- *Camada de Dados* Esta camada é responsável por armazenar os dados de entrada, e os dados de proveniência consumidos e gerados por cada etapa do *workflow* científico. Além disto esta camada tem informações caraterísticas do ambiente coletadas por um agente autônomo. Seus componentes estão instalados em máquinas virtuais específicas para armazenamento de dados.

2.3.1 Representação de *Workflows* Científicos

A representação das atividades e dos dados no SciCumulus é baseada em um modelo algébrico proposto por Ogasawara [38] para realizar a execução paralela de suas atividades e ativações. Segundo tal modelo os dados de um *workflow* são representados como relações e as atividades são regidas por operações algébricas.

O *workflow* é especificado por meio de um arquivo XML conforme o esquema XPDL. O SciCumulus possui um elemento raiz *SciCumulus*, tal elemento possui dois filhos o *database* que armazena as informações de conexão com o banco de dados e o *SciCumulusWorkflow* que descreve o *workflow* que se deseja representar. Cada *workflow* no SciCumulus é descrito pelos atributos:

- *Tag*: Define um rótulo para o *workflow*.
- *Description*: Representa a descrição que o cientista atribuiu para o *workflow* representado, servindo também como anotação para o *workflow*.
- *Exectag*: Serve para definir uma rodada do *workflow*, ou seja, definir a qual ensaio do *workflow* esta representação se refere.
- *Expdir* Define o diretório nas diversas máquinas virtuais onde se encontram os dados do experimento.

O *SciCumulusWorkflow* também serve como elemento agrupador no XML, pois ele é composto por diversas atividades, as *SciCumulusAcitities*. Uma *SciCumulusAcitity*, possui *tag* e *description* de forma análoga ao *SciCumulusWorkflow*, além desses campos, ela também possui:

- *Type* Define o tipo de operação algébrica que está sendo utilizado.
- *Activation* Define a linha de comando que deverá ser executada, ela deverá estar parametrizada para que em tempo de execução tais parâmetros sejam substituídos pelos seus valores reais.
- *Templatedir* Caso o programa necessite de um *script* para ser executado, o caminho deste *script* deve estar definido neste parâmetro.
- *Relation* Cada relação é composta de Campos (atributo *Field*) que definem os dados que são trafegados entre as atividades.

- *File* Representa os arquivos que devem ser manipulados durante a execução do *workflow*. inclusive os arquivos que devem ser instrumentados.

2.3.2 Modelo de Execução

Para a execução do *workflow* é adotada uma política *master-slave* onde o nó principal é responsável por gerenciar as ativações que serão executadas e persistir os dados de proveniência gerados por esta execução, enquanto os demais nós são responsáveis pela execução de cada ativação de fato.

Os nós executores podem ser multi-*threads*, ou seja, um nó executor pode executar N tarefas simultaneamente, para controlar esta execução em paralelo o *SciCumulus* utiliza dois principais conceitos, o *body* e a *head*.

A *head* é responsável pela execução das diversas ativações que um *workflow* possui, sua principal responsabilidade é executar o programa associado a uma dada ativação e capturar a proveniência de tal execução.

O *body* é responsável pelo gerenciamento do fluxo de execução de atividades, este possui diversas *heads*, e é responsável por as atribuir as ativações que serão executadas. De modo que é responsabilidade do *body* saber a situação atual de cada *head* e pedir ao nó *master* uma tarefa quando uma *head* estiver disponível.

O nó *master* também é capaz de executar atividades, porém em quantidade menor que os nós *slaves*, visto que o nó *master* também é responsável pelo escalonamento das atividades. Podemos ver um exemplo dessa arquitetura na Figura 2.4.

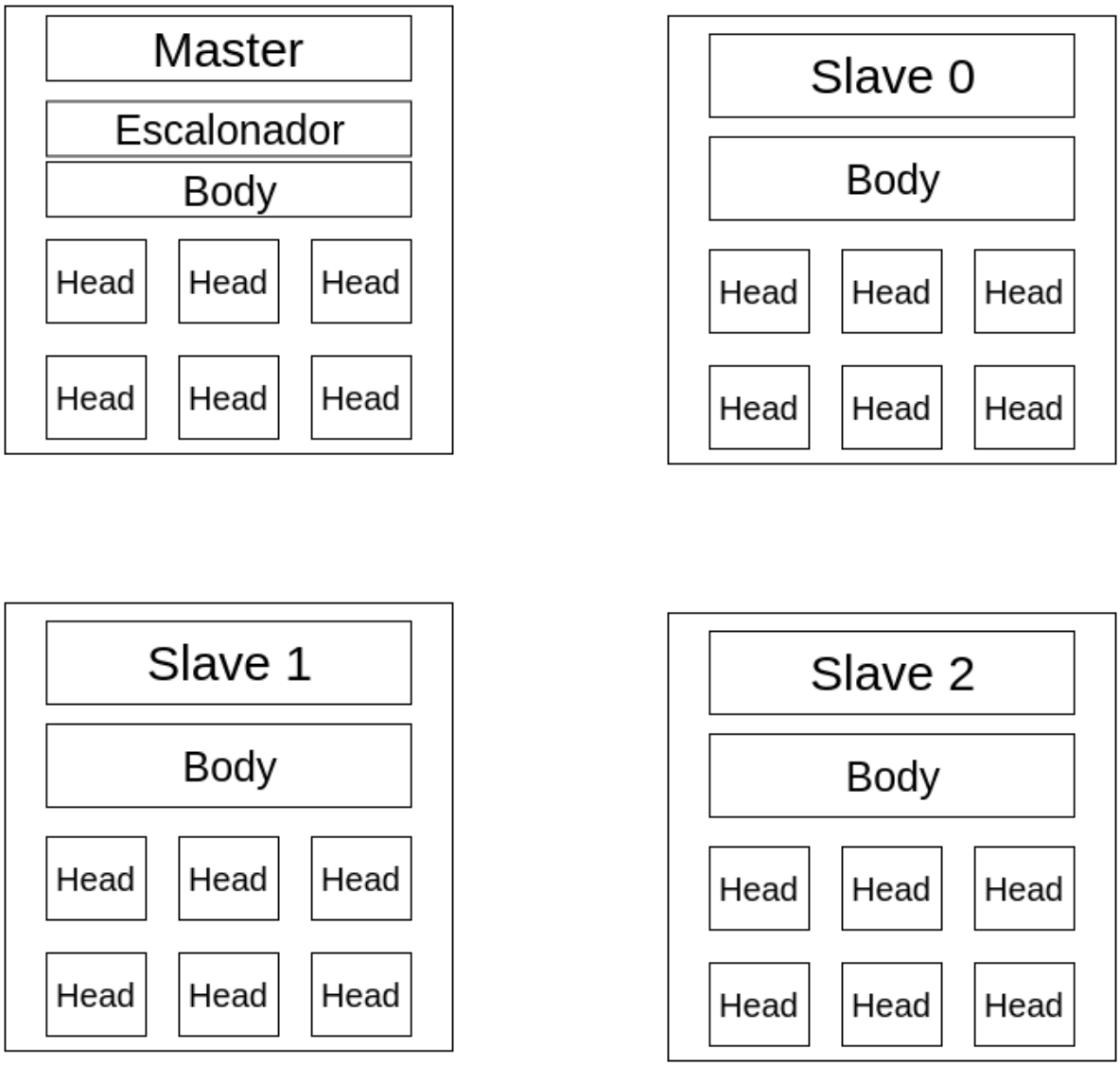

Figura 2.4: Arquitetura *Body* e *Head* do SciCumulus.

Um *workflow* como dito anteriormente, é composto por várias atividades, essas atividades podem possuir diversas ativações, dependendo dos parametros informados na execução, como exemplificado na Figura 2.5.

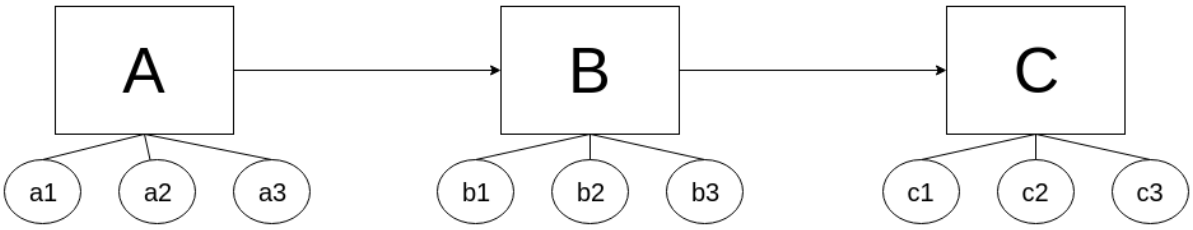

Figura 2.5: Exemplo de Workflow.

O SciCumulus provê dois tipos escalonadores dinâmicos, o DYN-FAF e o DYN-FTF, tais escalonadores são executados no nó *master* e são responsáveis por decidir qual será a atividade que será disponibilizada para o nó *slave* que está fazendo a requisição.

O DYN-FAF irá executar o *workflow* pensando nas atividades, ou seja, ele irá executar todas as ativações da atividade A antes de começar a executar alguma ativação da atividade B, como ilustrado na Figura 2.6.

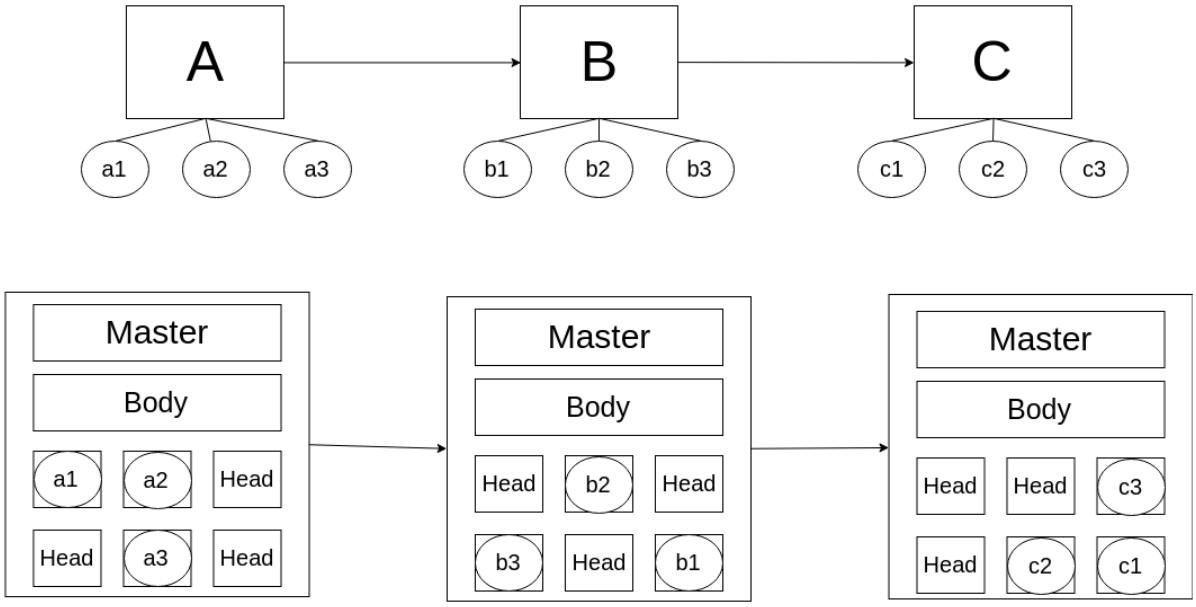

Figura 2.6: Exemplo de execução com o DYN-FAF.

Já o DYN-FAF irá separar a execução pelas ativações, ou seja, ele irá montar todo o encadeamento de atividades para uma combinação de parâmetros fornecidas pelo cientista e irá atribuir a uma *head*, e esta *head* será responsável por todo o fluxo de execução do *workflow* para este conjunto de parâmetros, como ilustrado na Figura 2.7.

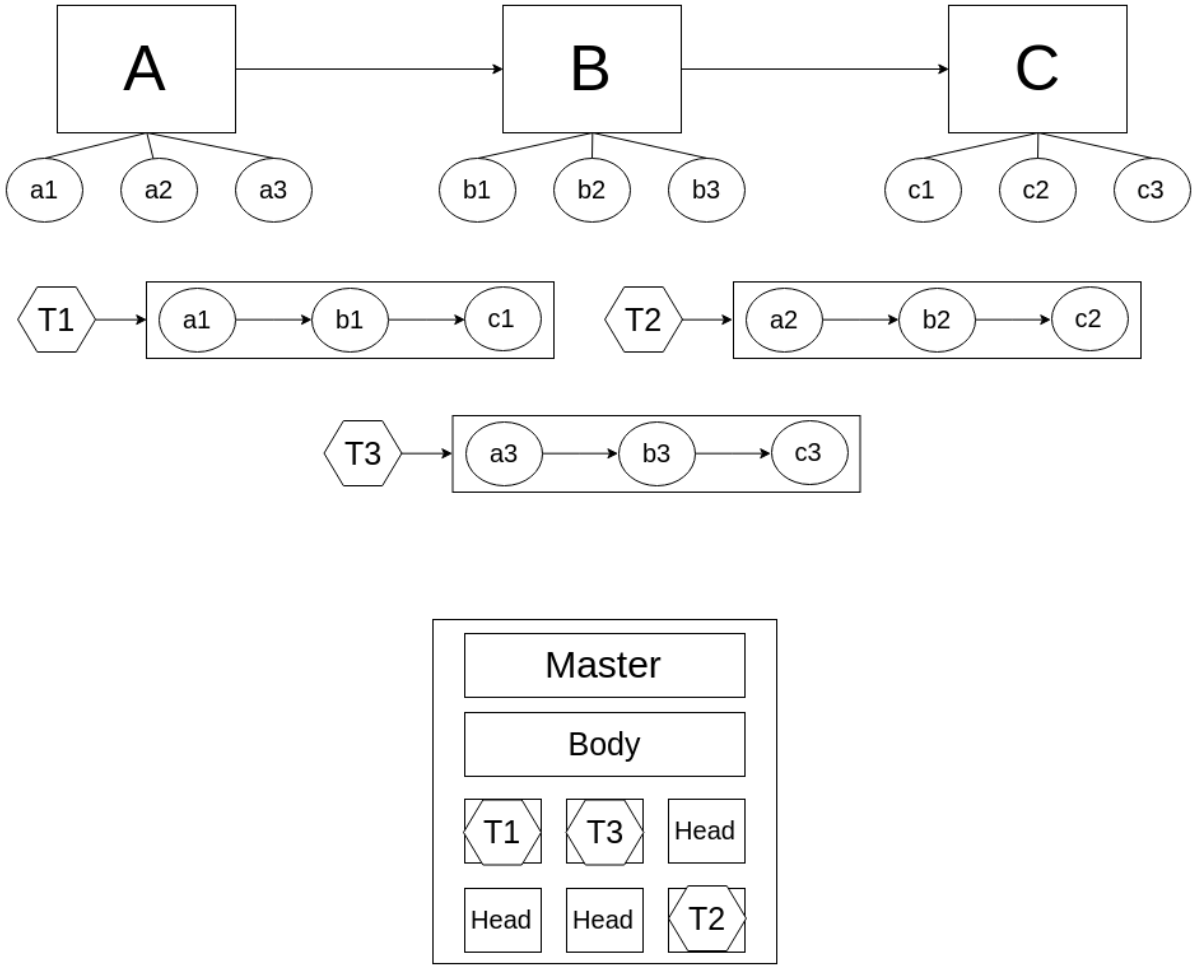

Figura 2.7: Exemplo de execução com o DYN-FTF.

2.4 GPGPU

As unidades de processamento gráficos, conhecidas como GPU (*Graphic Processor Unit*), surgiram com o propósito inicial de processar operações gráficas intensas para possibilitar a renderização de objetos e cenas complexas.

Mas olhando de uma forma mais ampla a GPU foi modelada para atender a uma classe de problemas com três principais características, e com o passar do tempo a comunidade identificou outras aplicações que se encaixavam em tais especificações [42], sendo elas:

- *Necessidade de amplo poder computacional.* Renderizadores em tempo real necessitam manipular bilhões de pixeis, e cada pixel necessita ser submetido a centenas de operações. A GPU necessita de um enorme poder computacional para atender essas demandas em tempo real.

- *Paralelismo é essencial* O *pipeline* gráfico é adequado para paralelismo. Operações em vértices e fragmentos são diretamente ligadas a unidades computacionais programáveis, de modo que cada vértice é representado por uma unidade computacional quando for necessário. O que pode ser aplicado em outras áreas da computação.

- *Vazão é mais importante que latência* O sistema visual humano opera em termos de milissegundos, enquanto um processador moderno opera em termos de nanosegundos, de modo que a latência de uma operação específica não é importante, e sim a quantidade de operações que podem ser feitas em paralelo, de modo que a GPU prioriza a vazão à latência, vários outros problemas possuem a mesma característica.

A alguns anos atrás as GPUs utilizavam funções fixas para renderização, o que era muito bom para renderizar gráficos 3D, mas não ia muito além disto. Desde então a GPU vem evoluindo, tanto em termos de *hardware*, quanto em termos de criação de plataformas de desenvolvimento de alto nível. Com este novo cenário outras aplicações que atendem aos requisitos citados acima podem utilizar GPUs para otimizar seu desempenho, mesmo que não tenha nada gráfico envolvido, a partir disto nasceu o conceito de GPU para propósitos gerais (do inglês *General Purpose Graphical Processing Unit*, ou GPGPU).

A diferença de programação entre CPU e GPU é conceitual, enquanto a CPU busca otimizar um código sequencial, a GPU foca em executar o mesmo código para diversos dados, tal modelo de programação é conhecido como único programa, múltiplos dados (do inglês *Single Program Multiple Data*, ou SPMD).

No cenário SPMD, os dados são independentes uns dos outros, e não se comunicam durante a execução do programa, de modo que cada um deles execute a código que lhe foi passado de maneira independente.

Códigos escritos nesse padrão são conhecidos como única instrução, múltiplos dados (do inglês *Single Instruction Multiple Data*, ou SIMD), tais códigos podem ser mais complexos, e exigir que existam ramificações. É possível criar ramificações em um ambiente GPU, porém existe uma punição alta para casos de ramificações incoerentes. Tais ramificações podem ser feitas no contexto de blocos, um bloco é um conjunto de *threads* a serem executadas em uma GPU, todas as *threads* do mesmo bloco devem executar o mesmo código, caso contrário a GPU irá executar todas as ramificações possíveis para todos os dados naquele bloco, logo o desempenho de uma aplicação feita em GPU está altamente ligada a forma como ela é mapeada para a GPU.

Capítulo 3

Abordagem Proposta

3.1 Arquitetura Proposta

Como discutido nos capítulos anteriores, os experimentos científicos *in silico* demandam um alto poder de processamento e manipulam uma grande quantidade de dados. Isto motivou a comunidade científica a procurar métodos para otimizar a execução de seus experimentos.

Por tal motivo diversas pesquisas vêm sendo realizadas para acelerar a execução destes experimentos, por meio de uso de grades computacionais e mais recentemente nuvens computacionais. Estes avanços possibilitam paralelizar as atividades de um *workflow*, o que aumenta sua sua vazão, porém alguns *workflows* possuem atividades demasiadamente pesadas, o que acaba criando um gargalo na execução do *workflow* como um todo.

Dada a natureza destes programas, eles podem se utilizar de *hardwares* aceleradores para otimizar o seu tempo de execução. Por isto ultimamente está havendo um esforço conjunto da comunidade para criar códigos utilizando tal abordagem e assim diminuir o gargalo criado por tais programas.

Um acelerador muito utilizado recentemente são as GPUs, e está havendo um esforço na criação de uma linguagem que consiga traduzir um determinado código para linguagens GPGPU, de modo que é possível um cientista escrever apenas um código e ter em mãos sua versão CPU e GPU sem nenhum custo adicional, além da possibilidade de traduzir códigos legados para uma linguagem GPGPU.

3.1.1 Representação de *Workflows* Científicos

Levando em conta este cenário onde o cientista possui em mãos diversas versões de um mesmo programa, onde cada versão é executada em um ambiente diferente, é interessante que o SGWfC consiga decidir qual versão utilizar em tempo de execução, tirando assim a responsabilidade dessa escolha do cientista, afinal ele não pode prever todos os possíveis cenários que podem acontecer em tempo de execução.

De forma que nossa proposta é que o cientista disponibilize todas as versões que ele possui do programa, e em tempo de execução o SGWfC decidir qual a melhor forma de executar este programa dado o cenário atual da execução do *workflow*, e dos recursos disponíveis no momento.

Esses programas podem apresentar restrições para sua execução, o que pode vir a impossibilitar sua execução em um determinado nó por falta de poder computacional. Um programa também pode pedir uma certa configuração do hardware para ser executado de forma ideal, apesar de sua execução ser possível em configurações inferiores do hardware de forma menos eficiente.

Essas restrições são fundamentais para uma boa execução do *workflow*, por exemplo, se uma atividade possuir uma versão para GPU, porém precisar de mais memória que o disponível, ela não deverá ser executada em sua versão GPU, mas em outra versão para manter a integridade da execução do *workflow*.

Para atender tais exigências foram criados requisitos para cada atividade criada no *workflow*, tais requisitos devem expressar as limitações ou exigências que uma atividade possui em cada ambiente dado, esses requisitos foram adicionados como propriedades ao *SciCumulus Workflow*, existem dois tipos de requisitos:

- **Requisitos Mínimos** São os requisitos mínimos que o *hardware* deve atender para executar tal atividade, caso o esses requisitos não sejam atendidos a atividade não será uma candidata a ser executada em tal *hardware*.
- **Requisitos Recomendáveis** São os requisitos que o cientista prefere que o *hardware* possua ao executar uma tarefa, porém se não os possuir o *hardware* pode a executar desde que cumpra os requisitos mínimos. Caso o *hardware* não atenda os requisitos recomendáveis a atividade entrará em sua lista de candidatos, mas só será executada nele se for a que o *hardware* melhor cumpre os requisitos recomendados.

Os requerimentos são informações opcionais, de modo que o SGWfC irá tentar exe-

cutar o *workflow* mesmo que o cientista não tenha os fornecido, porém se o programa de uma atividade possuir requisitos mínimos e isto não for informado, provavelmente a tarefa irá falhar e comprometer o desempenho de todo o experimento.

Essas novas propriedades foram pensadas para ampliar o modelo XML já presentes no SciCumulus como discutido na Seção 2.3.1.

3.2 Modelo de Proveniência

Uma das principais tarefas da proveniência é descrever o ambiente de execução do experimento, assim como o modo como o experimento interagiu com este ambiente, tal proveniência é conhecida como proveniência prospectiva.

A principio um programa para um SGWfC é uma caixa preta, ou seja, o SGWfC não sabe como este programa foi modelado para executar, nem os recursos que este necessita, de modo que pode-se argumentar que é trivial executar um programa feito para um acelerador específico em qualquer SGWfC apenas configurando a forma como este programa será invocado pelo SGWfC.

Porém um SGWfC não tem apenas a função de executar um *workflow*, mas também de capturar e disponibilizar os dados de proveniência para o cientista, este é o principal desafio para um SGWfC poder de fato executar um programa feito para um acelerador específico: como cole.

Sabemos que os aceleradores são feitos para suprir necessidades específicas, e que devido a isto as informações que devemos extrair deles para uma boa captura de proveniência não diferem somente da CPU, como também diferem entre si.

Deste modo para podermos de fato falar que um SGWfC suporta a execução de um acelerador qualquer, precisamos garantir que este capture os dados relevantes referentes a tal acelerador.

Para fazer isto precisamos ter um conhecimento grande sobre a forma como tal acelerador funciona e quais dados são relevantes ser extraídos do mesmo, e também ser possível extrair esses dados de uma execução de uma ativação do *workflow*.

Além disto precisamos armazenar tais dados no banco de dados, este, por sua vez, deve estar preparado para receber informações de proveniência de diversos tipos de aceleradores, que terão métricas diferentes.

Nesta seção iremos discutir não somente a ampliação no modelo de proveniência e banco de dados que foram necessários para adaptar o SciCumulus a este novo cenário, como também discutiremos as métricas extraídas da GPU Cuda e o motivo de cada uma delas.

3.2.1 Ampliação do modelo de proveniência

Para englobar as alterações propostas para o SciCumulus precisamos adaptar o PROV-Wf ao novo cenário de múltiplas formas de execução das atividades que compõe um *workflow* científico.

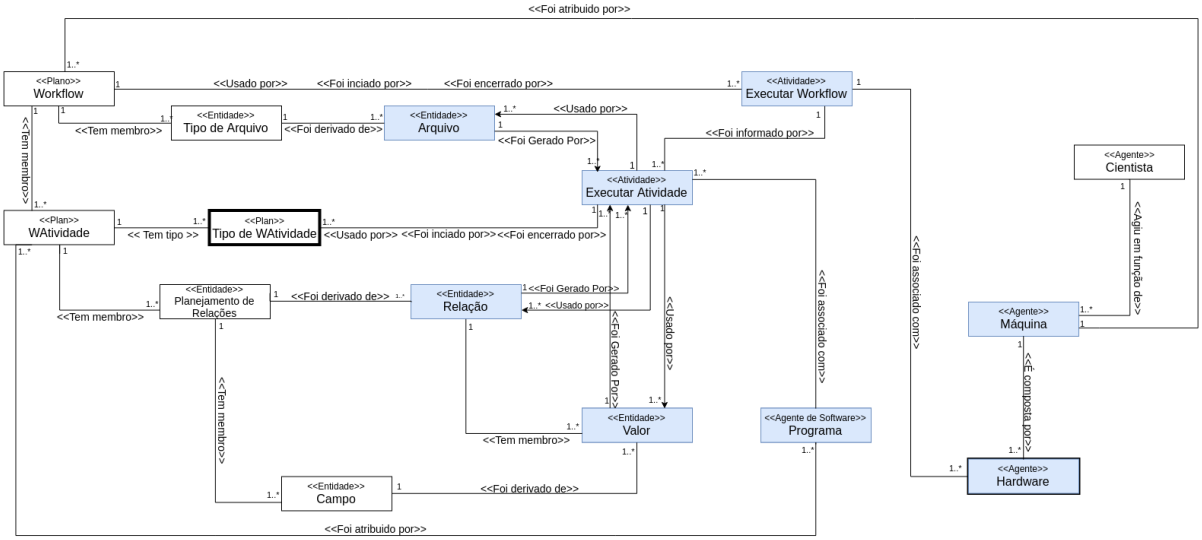

Figura 3.1: Ampliação do PROVWf.

Como podemos ver na Figura 3.1, foram adicionados três novos agentes ao modelo anterior do PROF-Wf, sendo eles:

- *Requisitos* Temos agora que uma atividade possui requisitos para poder ser executada em um determinado ambiente, portanto incluímos um novo [[quadrado]] para representar tal comportamento.

- *Tipo de WAtividade* No modelo anterior do PROV-Wf a WAtividade era um planejamento da atividade que seria executada, por isso era ligada diretamente com a atividade Executar Atividade.

  No novo cenário uma WAtividade se refere somente ao conceito científico da atividade, e agora esta WAtividade está associada a Tipo WAtividade, este se refere as múltiplas formas que tal atividade pode ser executada pelo SGWfC.

Portanto nesta nova versão do PROV-Wf o Tipo WAtividade que se relaciona com a atividade Executar Atividade. Uma vez que este é o planejamento da forma como a atividade será executada.

- *Hardware* O Hardware é o agente de execução do *workflow*. Anteriormente esta tarefa era associada diretamente ao agente Máquina. Porém agora existem tarefas que podem ser executadas em um Hardware e não em outro, portanto foi necessário aumentarmos o nível do detalhamento na representação do novo PROV-Wf.

3.2.2 Extração e Armazenamento de Dados de Aceleradores

Como discutido anteriormente um dos desafios de capturar a proveniência de aceleradores é identificar quais dados são interessantes serem armazenados para tal acelerador. Além disto, um dado pode ser importante para um tipo de acelerador, mas não fazer sentido para outro tipo. Por exemplo a versão CUDA que o programa foi executado é importante para GPUs da Nvidia, porém não faz sentido para outros aceleradores.

Com isto em mente temos o desafio de armazenar as informações relevantes de todos os aceleradores utilizados durante a execução do *workflow*. Para manter a expansibilidade do SGWfC para qualquer tipo de acelerador, definimos o banco como mostrado na Figura 3.2.

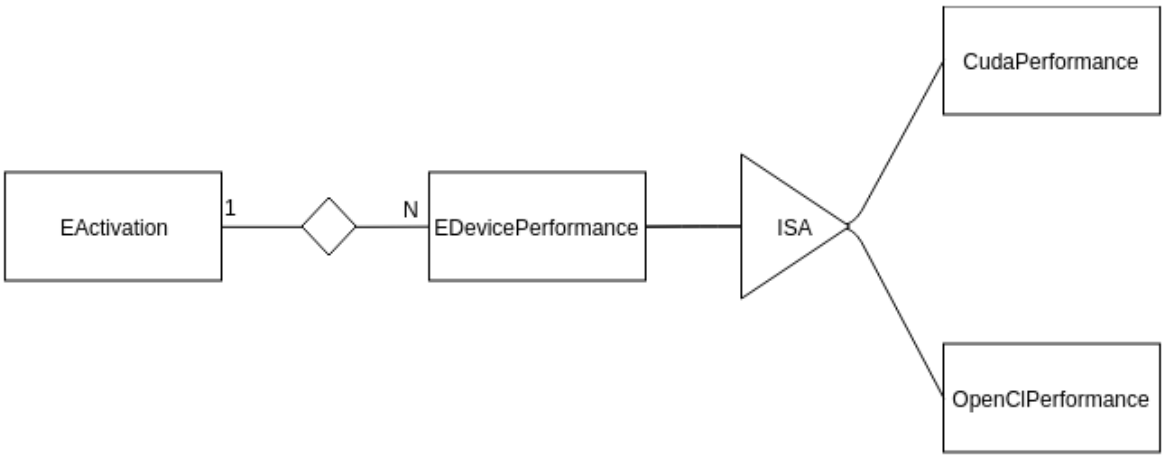

Figura 3.2: Proveniência de Múltiplos Aceleradores.

Atualmente o SciCumulus suporta GPUs Nvidia, para tal acelerador as métricas que são utilizadas são:

- *Versão do Driver* Capturado para informar qual versão CUDA a GPU que executou

a atividade utilizou.

- *Número Serial* Utilizado para detectar fisicamente a GPU que executou uma determinada tarefa, nem todas as GPUs oferecem esse recurso.
- *Percentual de Uso* Informa a quantidade de poder computacional foi utilizado em relação ao que a GPU oferece.
- *Memória Utilizada* Informa a quantidade de memória da GPU que foi utilizada utilizada.
- *Temperatura da GPU* Informa a temperatura que a GPU alcançou durante a execução da atividade.
- *Energia utilizada* Informa a energia utilizada pela GPU.

3.3 Execução do *Workflow*

Conforme mencionado na Seção 2.3.2, o fluxo de execução do SciCumulus é essencialmente composto pelo *body* e suas *heads*. Neste novo cenário, envolvendo unidades aceleradoras, especificamente *GPUs*, este modelo precisa receber alguns ajustes, para poder assim representar tais aceleradores.

Até então as *heads* apesar de serem independentes entre si, eram executadas em um mesmo ambiente, ou seja, se uma *head* podia executar uma tarefa as outras também poderiam.

Em um cenário de *GPU*, onde um computador possui uma unidade com N *CPUs* e M *GPUs*, as *heads* precisam passar a conhecer o ambiente que serão executadas, para que assim possa haver controle sobre qual ambiente estará sendo utilizado para cada tarefa, e qual estratégia de captura de proveniência deverá ser utilizada.

De modo que o fluxo de execução do *SciCumulus* precisou sofrer algumas mudanças na inicialização, no escalonador de tarefas e na captura de proveniência para este novo cenário, tais mudanças serão abordadas nesta seção.

3.3.1 Inicialização

O sistema de inicialização do SciCumulus tem como principal tarefa inicializar a comunicação entre as máquinas que iram executar o *workflow* científico, e detectar o potencial

de cada máquina que irá executar os diversos programas que compõe o *workflow*.

Até então o SciCumulus apenas é capaz de operar devidamente em ambientes que possuem um ambiente puramente composto por CPUs, de modo que na inicialização era necessário apenas medir a capacidade de computação de cada máquina e estabelecer a comunicação entre elas.

Neste modelo cada máquina precisa detectar se a mesma possui algum acelerador que o SciCumulus possua suporte, neste cenário é necessário ter conhecimento da arquitetura de tal acelerador, para que assim seja extraído o conhecimento necessário para determinar o potencial computacional de tal acelerador.

Na inicialização também são inicializados o *body* e suas diversas *heads*, neste cenário as *heads* passam a possuir tipos, de modo que as *heads* agora diferem em relação ao escopo do programa que poderá ser executado nela.

Detecção de Aceleradores

```
Funcao CapturaInformacoesDaMaquina()
    IAceleradorSuportado aceleradoresSuportados;
    aceleradoresSuportados = new Lista<IAceleradorSuportado>();
    Para(cada acelerador em aceleradoresSuportados)
        Se(acelerador.estaPresenteNaMaquina())
            AceleradorInfo aceleradorInfo;
            aceleradorInfo = acelerador.capturaInformacoesTecnicas();
            Maquina.Aceleradores.adiciona(aceleradorInfo);
        FimSe
    FimPara
FimFuncao
```

Como podemos observar no pseudo-código acima o SciCumulus varre a máquina procurando pelos aceleradores que o mesmo possui suporte. Caso encontre algum ele captura as informações relevantes daquele hardware e o persiste para que durante a execução as atividades saibam a configuração dos aceleradores daquela máquina.

3.3.2 Escalonador de Tarefas

Tendo agora um cenário onde se possui *heads* com especializações, o papel de cada componente sofreu diversas mudanças.

O *body*, componente que controla o fluxo de execução localmente, agora precisa controlar também o tipo da *head* disponível para execução, e assim informar para a máquina *master* o tipo do programa que é elegível para tal *head*.

A *head* por sua vez é responsável não somente pela execução do experimento, mas também pela captura de sua proveniência, de modo que a *head* passa a ter a responsabilidade de saber capturar a proveniência para o acelerador que está representando.

No nó *master* a escolha da ativação que será escolhida para execução foi alterada, visto que neste cenário o nó *master* deve fornecer uma ativação que tenha um modo de execução no ambiente que lhe foi passado pelo *body* da máquina *slave*.

Neste novo cenário cada atividade pode possuir diversos modelos de execução, como apresentado na Figura 3.3 com o workflow SciPhy que foi usado na avaliação experimental e será detalhado na Seção 4.

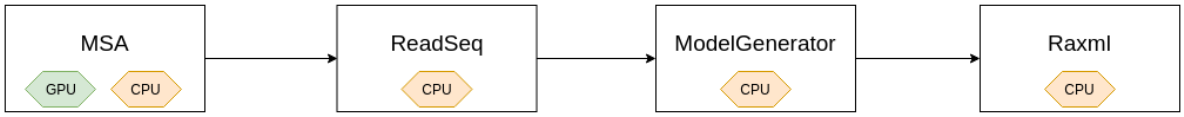

Figura 3.3: Exemplo de Modelagem Híbrida

De modo que temos um novo cenário, onde uma tarefa pode não ser elegível para uma *head*, no caso da tarefa não possuir um modelo de execução para o ambiente que tal *head* representa. Com isto a escolha do modelo de escalonamento que será utilizado no nó *master* assume uma importância maior no tempo que um experimento levará para ser concluído.

No pseudo-código abaixo representamos o fluxo de execução da escolha da atividade que será escolhida para ser executada. O nó executor irá pedir para o nó *master* uma atividade, e irá passar como parâmetro a sua *head* que está disponível.

Por sua vez o nó *master* irá verificar o tipo de *hardware* associado aquela *head* e irá pegar as atividades que podem ser executadas naquele *hardware*.

Após isto ele irá remover as atividades que o *hardware* não possui condições de executar e ordenar as atividades pela quantidade de requisitos recomendados da atividade que o *hardware* atende, e irá retornar a atividade que o *hardware* mais atende aos requisitos recomendados.

Escalonamento de Atividades

```
Funcao escolheAtividade(Head headDisponivel){
    TipoHardware hardware = headDisponivel.pegaTipoDoHardware();
    Lista<Ativacao> ativacoes = pegaAtividadesExecutaveisEm(hardware);
    Se tiverAtivacoes
        Escalonador escalonador = hardware.escolheEscalonador();
        escalonador.filtra(hardwareCumpreRequisitosMinimos());
        escalonador.ordena(hardwareCumprerequisitosRecomendados())
        Ativacao ativacaoEscolhida = Ativaescalonador.pegaPrimeiro();
        retorna ativacaoEscolhida;
    FimSe
    Senao
        retorna vazio;
    FimSenao
FimFuncao
```

Como discutido anteriormente o SciCumulus provê dois modelos de escalonamento dinâmico de tarefas, o DYN-FAF e o DYN-FTF.

O DYN-FAF sofreu poucas mudanças, ele agora só precisa se preocupar com o tipo da *head* e da atividade, para atribuir corretamente as ativações as *heads* que são capazes de a executar, como mostrado na Figura 3.4.

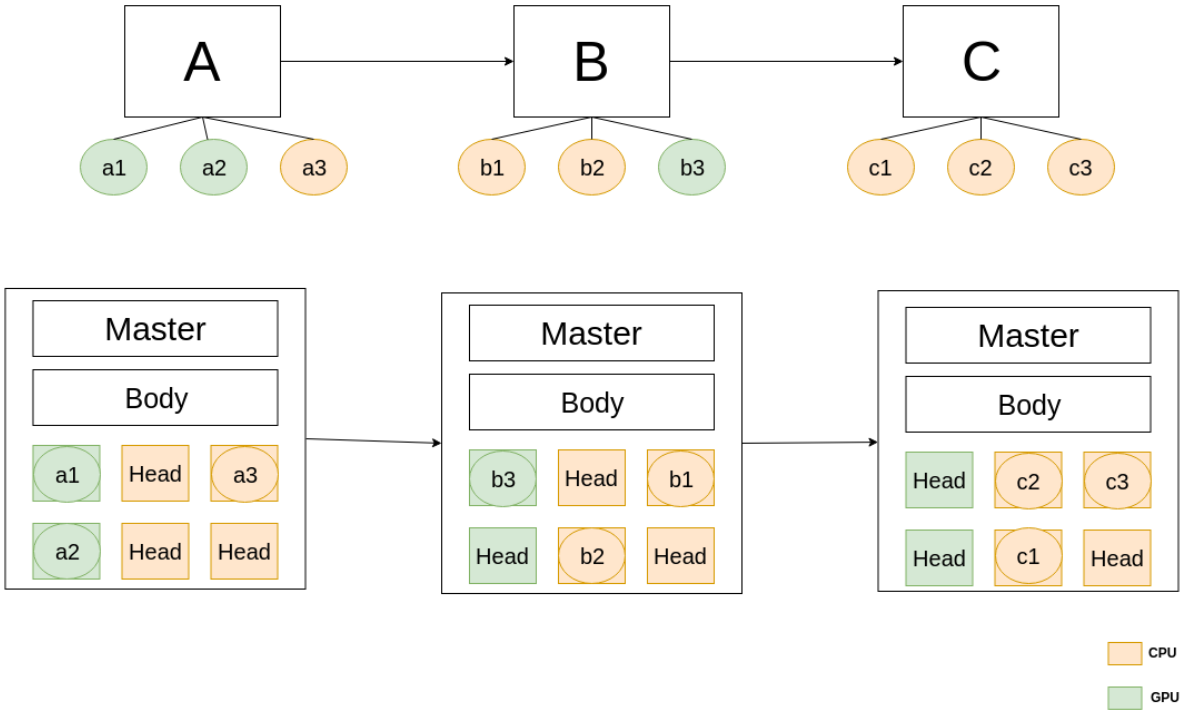

Figura 3.4: Exemplo Escalonamento FAF

Já o DYN-FTF se torna impraticável, pois ele atribui toda execução de uma combinação de parâmetros do *workflow* a uma única *head*, com o novo cenário onde uma *head* e as atividades possuem um tipo específico, essa *head* poderá não ser capaz de executar todo o fluxo do *workflow*, como mostrado na Figura 3.5, onde nenhum fluxo pode ser executado por nenhuma *head* por possuir formas de execução heterogêneas..

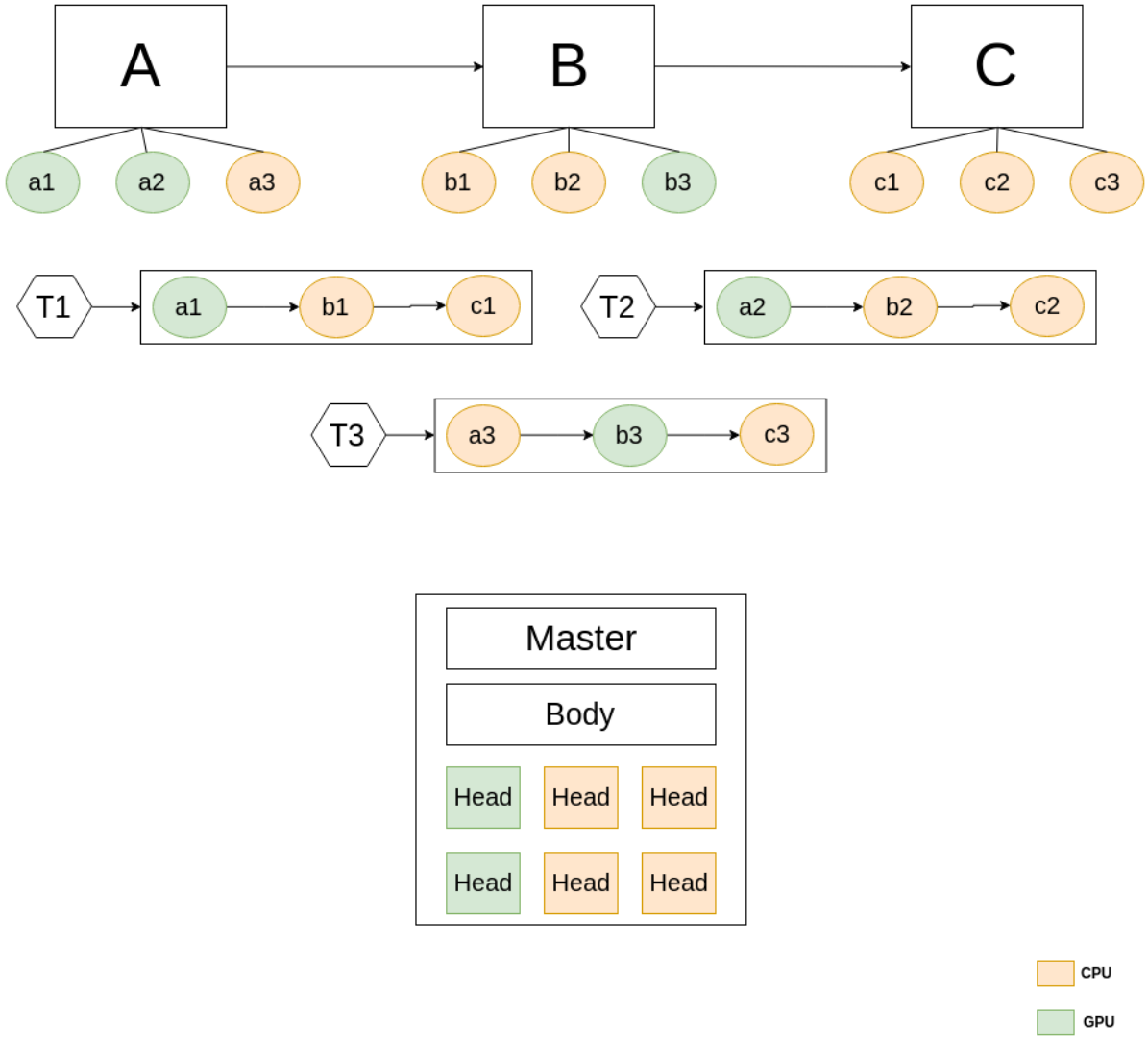

Figura 3.5: Exemplo Escalonamento FTF

3.3.3 Captura de Proveniência

Como discutido anteriormente a *head* possui a tarefa de capturar a proveniência da ativação que foi executada na mesma, em CPU ela faz isso através do comando sar, porém tal comando não funciona para qualquer tipo de acelerador.

De modo que foi necessária uma abordagem diferente para a captura de dados no ambiente GPU Cuda. Quando a *head* começa a execução de um programa, em paralelo ela abre uma *thread* que terá a única responsabilidade de consultar a situação atual da GPU em que tal ativação está sendo executada.

Nesta *thread* é constantemente executado um comando que captura a situação dos dados discutidos na seção 3.2.2. Após a execução é aplicada a média entre os valores

coletados e temos o consumo médio daquele da GPU daquele nó por uma ativação.

3.4 Modelo de Custo

Nesta seção iremos discutir sobre o modelo de custo do SciCumulus usado para o escalonamento de ativações e as adaptações feitas no mesmo para atender a execução de um *workflow* em um ambiente com GPU.

Este modelo é composto pelo tempo de execução do *workflow* e o custo de execução do mesmo, e será utilizado na hora do escalonamento de tarefas. O modelo de custo original do SciCumulus foi prosposto por Oliveira *et al.* [16].

O custo total da execução de um *workflow* é definido pela Equação 1:

$$Cost(W, P_L) = w_t * T_n(W, P_L) + w_m * (F_n(W, P_L)) \tag{1}$$

Onde w_t e w_m são os pesos atribuídos pelo cientista para priorizar, respectivamente, o tempo de execução ou o custo total da execução de modo que $w_t + w_m = 1$. As funções $T_n(W, P_L)$ e $F_n(W, P_L)$ são valores normalizados que representam o tempo de execução e o custo da execução respectivamente.

Podemos de forma similar definir a execução de uma ativação ac_i em uma máquina virtual vm_j como representado na Equação 2.

$$Cost(ac_i, vm_j) = w_t * T_n(ac_i, vm_j) + w_m * F_n(ac_i, vm_j) \tag{2}$$

3.4.1 Tempo de Execução

Nesta seção iremos discutir como estimar o tempo de execução de um *workflow* W. Temos um conjunto de ativações $AC = (ac_1, ac_2, ..., ac_n)$. O tempo de execução de uma ativação ac_i em uma máquina virtual vm_j é dado por $P(ac_i, vm_j)$. Considere também que o plano de provisionamento P_L define o conjunto de máquinas virtuais que irá compor o cluster virtual V_C, ou seja, $P_L = (vm_1, vm_2, ..., vm_{pl})$ e $P_L \subseteq VM$. Portanto definimos a execução de todas as ativações AC em um *workflow* W como $T(W, P_L)$. Por tal motivo o tempo de execução normalizado T_n utilizado na Equação 1 pode ser definido pela Equação 3.

$$T_n(W, P_L) = \frac{T(W, P_L)}{D_T} \tag{3}$$

Onde D_T é o tempo em que o usuário deseja executar W. Ambos D_T e D_F (custo financeiro desejado) são setados pelo usuário. Note que combinações destes valores podem resultar em cenários onde é impraticável a execução de W. Nós tomamos D_T e D_F em consideração no modelo de custo, porém os valores reais podem variar dependendo da execução real de W.

Para executar W é necessário inicializar as máquinas virtuais e executar os programas nelas. O tempo de inicialização é definido pelo tempo necessário para preparar as máquinas virtuais para execução de W. Ou seja, o tempo para inicializar a máquina virtual, instalar os programas necessários e preparar os parâmetros de configuração da máquina virtual. Desta forma o tempo total necessário para executar W é definido pela fórmula 3.4.2.

$$T(W, P_L) = I_T(P_L) + (e_{gpu} \times E_{T_{cpu}}(W, P_L) + e_{gpu} \times E_{T_{gpu}}(W, P_L)) \tag{4}$$

$I_T(P_L)$ representa o tempo para inicializar o ambiente para um plano de execução P_L, $E_{T_{cpu}}$ e $E_{T_{gpu}}$ é o tempo de execução dos programas executados por W utilizando o plano de execução P_L, e_{gpu} varia entre 0 e 1, e informa se foi utilizado ou não GPU. O tempo de inicialização das máquinas virtuais é definido pela Equação 5.

$$I_T(P_L) = m \times A_{I_T} \tag{5}$$

A_IT representa o tempo médio para inicializar uma máquina virtual. O valor de A_IT pode ser configurado pelo usuário de acordo com o ambiente utilizado. Que pode ser obtido pela média do tempo da iniciar, instalar os programas necessários e realizar as devidas configurações diversas máquinas virtuais. No restante desta dissertação iremos assumir que o plano de execução P_L utiliza $|P_L| = m$ máquinas virtuais e que apenas uma máquina virtual pode ser inicializada em um ambiente *web* por vez.

Assumindo que o plano de execução P_L corresponde ao instanciamento de N_cpu CPUs virtuais para executar W., de acordo com a lei de Amdahl o tempo de execução pode ser definido pela Equação 6.

$$E_{T_{cpu}}(W, P_L) = \frac{\frac{\alpha}{N_{cpu}} + (1-\alpha) \times W_l(W, I_D)}{C_{spc}} \tag{6}$$

Onde α representa a porcentagem da aplicação W_l que pode ser executada em paralelo. α pode ser obtido pelo tempo de execução de SWf consumindo uma pequena quantidade de dados I_D repetidamente utilizando quantidades diferentes de CPUs virtuais. Podemos definir $E_{T_{gpu}}(W, P_L)$ de forma análoga. Por exemplo, digamos que uma aplicação foi executada no tempo t_1 utilizando n_{cpu} CPUs virtuais, e t_2 para m_{cpu} CPUs virtuais, podemos obter α utilizando a Equação 7.

$$\alpha = \frac{m_{cpu} \times n_{cpu} \times (t_2 - t_1)}{m_{cpu} \times n_{cpu} \times (t_2 - t_1) \times n_{cpu} \times t_1 - m_{cpu} \times t_2} \tag{7}$$

C_{spc} representa a capacidade de performance média de cada CPU virtual, a qual é medida por FLOPS, operações de ponto flutuante, do inglês *Float-point Operations*. Podemos utilizar a equação 8 a unidade de F_{eq} é GHz e a de C_{spc} é GFLOPS.

$$C_{spc} = 4 \times F_{eq} \tag{8}$$

Finalmente, W_l representa a carga de W com quantidades especificas de dados I, que pode ser medida pelo número de FLOP. Todos os parâmetros α, W_l e C_{spc} devem ser configurados pelo usuário de acordo com as especificações da nuvem e de W. Nós calculamos o valor de da carga de trabalho de W utilizando a Equação 9 onde ac_j é uma ativação de W e w_l representa a carga de trabalho da ativação ac_j, consumindo a entrada definida por $entrada(ac_j)$.

$$W_l(W, I) = \sum_{ac_j \in W} w_l(ac_j, entrada(ac_j)) \tag{9}$$

3.4.2 Custo Financeiro

Nesta seção iremos definir a formalização usada para estimar o custo financeiro de um plano de execução P_L. A valor normalizado F_n utilizado na equação 1 pode ser definida pela Equação 10.

$$F_n(W, P_L) = \frac{F(W, P_L)}{D_F} \tag{10}$$

Onde F é o custo associado com a execução de W utilizando as máquinas virtuais definidas por P_L, D_F representa o custo financeiro informado pelo usuário.

Vamos assumir que cada ativação tem uma carga de trabalho $W_l(act_j, input(ac_j))$ similar a uma previamente definida. De modo similar a fórmula para estimar o tempo de execução, o custo financeiro também possui duas partes, a inicialização e a execução de W, como definido na Equação 11.

$$F_N(W, P_L) = I_{fc}(P_L) + E_{fc}(W, P_L) \tag{11}$$

Onde I_fc representa o custo financeiro associado ao fornecimento das máquinas virtuais utilizadas para executar W e E_fc é o custo financeiro para efetivamente executar W.

O custo financeiro para inicializar o ambiente de execução é estimado pela equação 12, isto é, a soma do custo de fornecimento de cada máquina virtual.

$$I_{fc}(P_L) = \sum_{ac_j \in W}^{m} (F(vm_i, s) \times \frac{(m-i) \times I_T}{T_Q}) \tag{12}$$

$F(vm_i)$ é o custo financeiro para se utilizar uma máquina virtual vm_i por um quantum de tempo em um site s. I_T representa a média de tempo do fornecimento de uma máquina virtual previamente definida. T_Q representa o quantum de tempo da nuvem, o que é a menor quantidade de tempo usada para calcular o custo do uso da máquina virtual.

O parâmetro m representa que m máquinas virtuais devem ser instanciadas para executar W. De forma similar ao calculo do tempo de execução iremos considerar que apenas uma máquina virtual pode ser inicializada por vez em um domínio *web*.

O custo financeiro pela execução de W é estimado pela equação 13. $E_T(SWf, P_L)$ é definido na equação 6. O parâmetro F_{cpc} representa o custo se utilizar um núcleo virtual de CPU em um quantum de tempo da nuvem, tal valor pode ser obtido pela divisão do preço do quantum de tempo da nuvem pela quantidade de núcleos virtuais. T_Q representa o quantum de tempo da nuvem.

$$E_{fc}(W, P_L) = n \times F_{cpc} \times \left\lfloor \frac{E_T(W, P_L)}{T_Q} \right\rfloor \tag{13}$$

Capítulo 4

Avaliação Experimental

4.1 Sciphy

O Sciphy é um workflow cientifico que foi projetado para gerar árvores filogenéticas com máxima verossimilhança. Ele foi projetado inicialmente para trabalhar com sequências de aminoácidos, podendo ser estendido para outros tipos de sequências biológicas.

Ele é composto por quatro atividades, sendo elas:

- *MSA* Constrói alinhamentos individuais. Ele recebe um arquivo multi-fasta como entrada, produzindo como saída um alinhamento (MSA). Neste ponto o SciPhy obtém o alinhamento individual.
- *ReadSeq* Converte o alinhamento individual para o formato PHYLIP.
- *Model Generator* Nesta atividade cada MSA é testado para encontrar o melhor modelo evolutivo.
- *RaXml* Nesta atividade tanto o modelo evolutivo quanto o MSA são utilizados para gerar árvores filogenéticas para cada um dos programas de MSA que foram eleitos.

Como os cientistas não conhecem *a priori* qual o método de alinhamento que produz o melhor resultado final, eles precisam executar o SchiPhy várias vezes, uma para cada método MSA. Estas atividades, respectivamente, executam as seguintes aplicações de bioinformática: programas de alinhamento genético (permitindo ao cientista a escolher entre o MAFFT, o Kalign, o ClustalW, o Muscle, ou o ProbCons), o ReadSeq [26], o ModelGenerator [31], o RAxML [50], um script Perl, o ModelGenerator e o RAxML. A Figura 4.1 apresenta a visão conceitual do SciPhy.

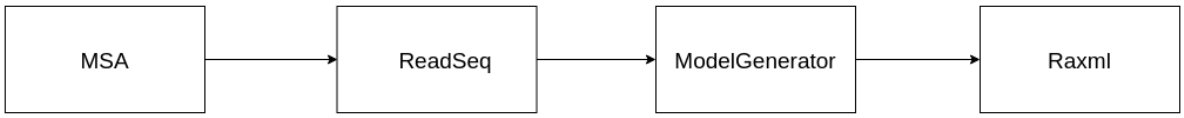

Figura 4.1: Sciphy - Modelo de Execução

A primeira atividade do SciPhy (MSA) constrói alinhamentos individuais utilizando um dos cinco programas disponíveis para alinhamento genético: o ClustalW, o Kalign, o MAFFT, o Muscle, ou o ProbCons. Cada programa de alinhamento recebe um arquivo multi-fasta (que pode representar um aminoácido ou não) como entrada (a partir de um conjunto de arquivos multi-fasta existentes), produzindo como saída um alinhamento (MSA). Neste ponto, o SciPhy obtém o MSA individual (produzido pelo programa de alinhamento eleito para a atividade). Na segunda atividade, o alinhamento é convertido para o formato PHYLIP [22]. Formato este que é alcançado através da utilização do programa ReadSeq. Cada MSA é testado na terceira atividade (Eleição do Modelo Evolutivo) para encontrar o melhor modelo evolutivo utilizando o ModelGenerator e ambos (MSA individual e modelo evolutivo) são utilizados na quarta atividade (Geração da Árvore Filogenética) para gerar árvores filogenéticas utilizando o RAxML com parâmetro de bootstrap configurável.

Para nosso experimento iremos utilizar a tarefa MSA com uma versão de GPU, para tal usaremos o CSMA [9], onde temos a opção de apenas trocando argumentos utilizar a versão CPU ou GPU da atividade MSA.

4.2 Experimentos

4.2.1 Ambiente

Para realizarmos os experimentos montamos um ambiente no AWS EC2, utilizando as máquinas G3 que são próprias para execuções de alto desempenho em GPUs.

Cada instância possui GPUs NVIDIA Tesla M60, cada uma com até 2.048 núcleos de processamento paralelo, 8 GiB de memória de GPU com suporte a OpenGL, DirectX, CUDA, OpenCL e Capture SDK. Além disto disponibilizam até 64 vCPUs baseadas em processadores Intel Xeon E5 2686 v4 2,7 GHz personalizados e 488 GiB de memória DRAM de host. A Amazon oferece 4 tipos de máquinas virtuais para as instâncias G3 que estão listadas na tabela abaixo.

Tabela 4.1: Experimentos Realizados

Modelo	CPUs	GPUs	Mem (GiB)	Mem GPU (GiB)
g3s.xlarge	4	1	30,5	8
g3s.4xlarge	16	1	122	8
g3s.8xlarge	32	2	244	16
g3s.16xlarge	64	4	488	32

4.2.2 Execução

Como discutido nos capítulos anteriores, o SciCumulus provê dois tipos de escalonamentos dinâmicos, o FAF e o FTF, porém o FTF assume que uma *head* é capaz de executar quaisquer atividade do *workflow*, o que não é mais verdade.

De forma que podemos analisar apenas o FAF, tal escalonamento executa todas as tarefas de uma atividade antes de começar a executar a próxima, com isso queremos analisar dois comportamentos. O tempo de execução conforme o número, e qualidade de GPUs aumenta, e a diferença na execução em um modelo apenas informando a execução GPU da atividade MSA e informando a versão GPU e CPU.

Para o primeiro experimento iremos utilizar apenas a versão GPU da atividade MSA, como ilustrado na figura 4.2, visto que queremos analisar o desempenho variando a quantidade e qualidade das GPUs. Os experimentos que serão executados estão listados na tabela 4.2.

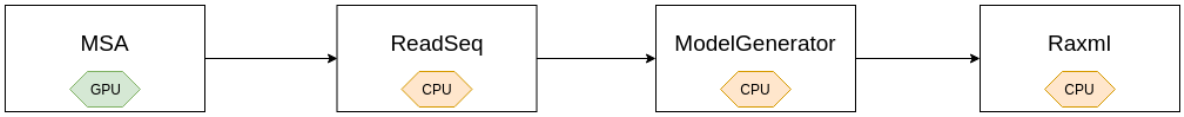

Figura 4.2: Sciphy - Modelagem

Para verificarmos a eficácia de se utilizar *heads* CPUs em adição as GPUs na primeira atividade do experimento iremos executar novamente na máquina g3s.16xlarge o experimento com 1000 ativações, porém utilizando o modelo ilustrado na figura 4.3.

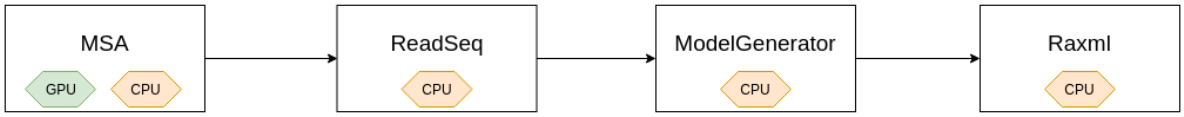

Figura 4.3: Sciphy - Modelagem (CPU & GPU)

Tabela 4.2: Experimentos Realizados

Quantidade de Ativações	Máquina Virtual	Método de Escalonamento
10	g3s.xlarge	FAF
10	g3s.4xlarge	FAF
10	g3s.8xlarge	FAF
10	g3s.16xlarge	FAF
100	g3s.xlarge	FAF
100	g3s.4xlarge	FAF
100	g3s.8xlarge	FAF
100	g3s.16xlarge	FAF
1000	g3s.xlarge	FAF
1000	g3s.4xlarge	FAF
1000	g3s.8xlarge	FAF
1000	g3s.16xlarge	FAF

4.2.3 Resultados

Nosso principal objetivo com tais experimentos é verificar a escalabilidade da nossa solução utilizando os recursos disponibilizados pelas instâncias G3 da Amazon, os resultados das execuções podem ser verificados na tabela 4.3.

Tabela 4.3: Experimentos Realizados

Ativações	Máquina Virtual	Tempo - MSA	Tempo Experimento
10	g3s.xlarge	00:00	00:04
10	g3s.4xlarge	00:00	00:03
10	g3s.8xlarge	00:00	00:02
10	g3s.16xlarge	00:00	00:02
100	g3s.xlarge	00:03	00:44
100	g3s.4xlarge	00:02	00:18
100	g3s.8xlarge	00:01	00:12
100	g3s.16xlarge	00:01	00:11
1000	g3s.xlarge	00:29	04:38
1000	g3s.4xlarge	00:25	02:54
1000	g3s.8xlarge	00:20	02:21
1000	g3s.16xlarge	00:18	01:44

Podemos observar que nossa atividade com versão GPU, a MSA, tem uma melhoria quando executadas em máquinas com GPUs mais potentes, porém podemos observar que houve pouco ganho nesta atividade quando realizamos o upgrade da máquina *g3s.8xlarge* para a *g3s.16xlarge*. Isto ocorre por estarmos utilizando várias entradas pequenas, não aproveitando todo o potencial do *hardware* oferecido pela *g3s.16xlarge*.

Além disto essa melhoria pode apenas ser observada de fato quando o número de

ativações é significativo, pois para uma quantidade de ativações muito pequenas o tempo de processamento é praticamente irrelevante para o MSA. Como observado na figura 4.4.

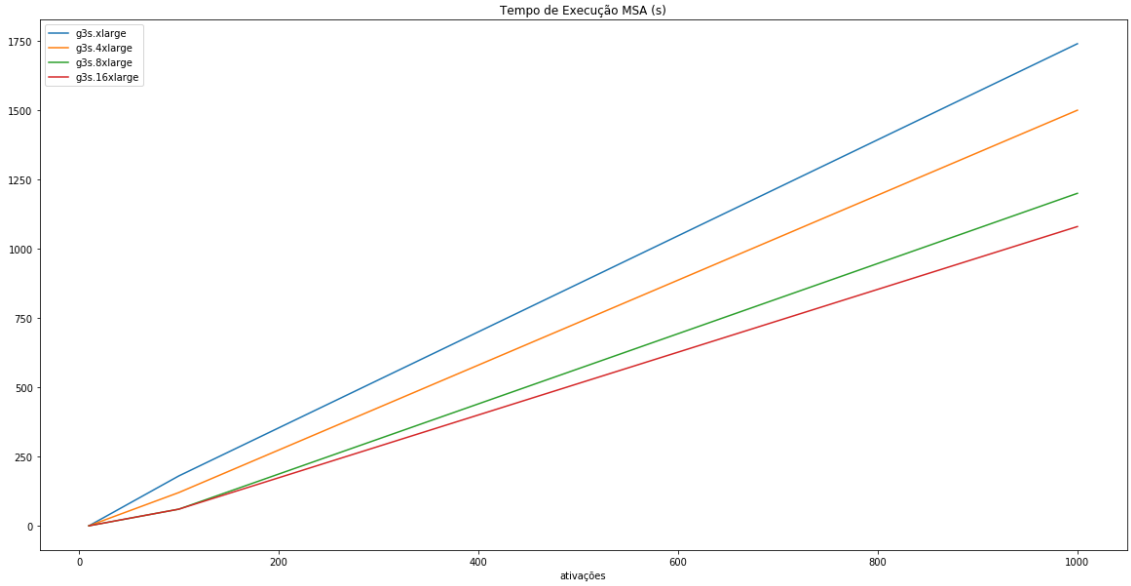

Figura 4.4: Tempo de Execução MSA

O *workflow* como um todo é executado de forma bem mais rápida na *g3s.16xlarge* em grande parte pela atividade *modelgenerator* que é muito custosa, e acaba se aproveitando melhor o aumento das vCPUs que temos conforme melhoramos a máquina. Como observado na figura 4.5.

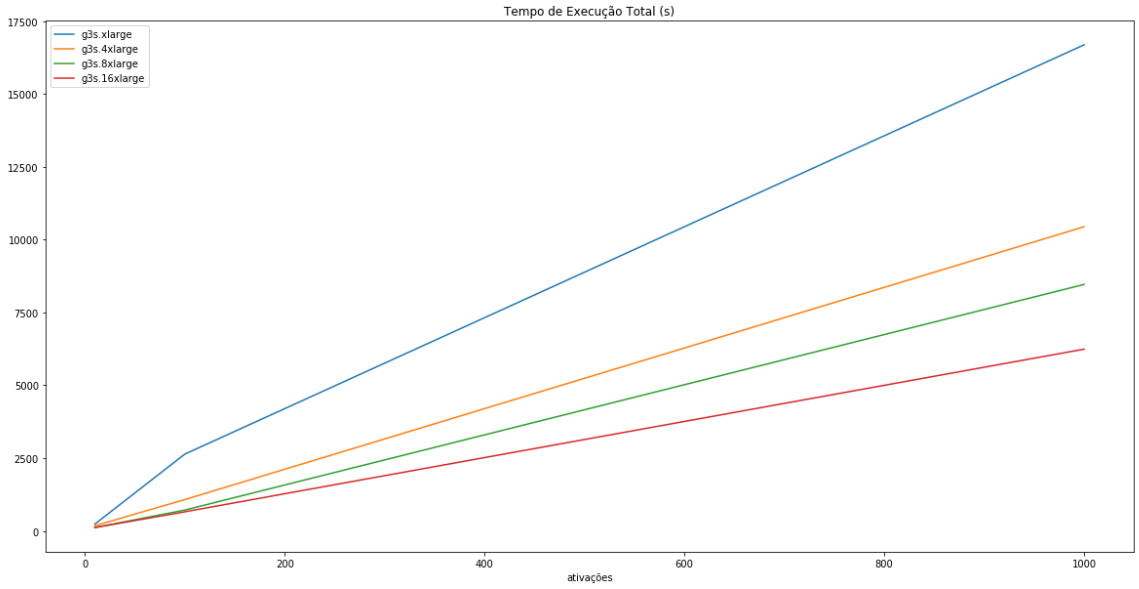

Figura 4.5: Tempo de Execução Total

Outro ponto que foi amplamente discutido nesta dissertação foi a possibilidade do cientista disponibilizar múltiplas versões de uma atividade para o *workflow* por si só saber lidar com esta informação e não ficar com *hardware* ocioso durante a execução de uma atividade com versão para GPU.

Pensando nisto montamos o experimento 4.3 para podermos comparar com a execução realizada somente com a atividade MSA disponivel em sua versão GPU. Os resultados deste experiemnto estão disponíveis na tabela 4.4.

Tabela 4.4: Experimentos - MSA (CPU/CPU e GPU)

Ativações	**Máquina Virtual**	**Tempo - MSA**	**Tempo Experimento**
1000	g3s.16xlarge - GPU	00:18	01:44
1000	g3s.16xlarge - CPU e GPU	00:10	01:34

Podemos notar que ao disponibilizar a versão CPU para o SciCumulus ele foi capaz de utilizar os recursos CPU disponíveis para executar algumas tarefas nas *heads* CPU o que agilizou o tempo total em que ele executou todo o *workflow*, mesmo que estas não executem tão rápido quanto as GPUs, elas o fazem em paralelo aumentando a vazão de ativações realizadas.

Portanto a funcionalidade de permitir múltiplas formas de ativação para uma atividade pode de fato agilizar a velocidade com o que o experimento é concluído, porém devemos ter cuidado, pois apesar do MSA GPU utilizar majoritariamente recursos GPU, ele também necessita de recursos CPU, portanto uma má configuração em relação a número de *heads* por hardware pode levar a uma disputa por recurso, atrasando a execução do experimento, ao invés de acelera-la.

Capítulo 5

Trabalhos Relacionados

A performance da execução de um experimento científico vêm sendo constantemente colocada como um desafio da área, isso se dá em grande parte ao crescimento da massa de dados que os experimentos atuais estão consumindo.

Devido a isto a comunidade se voltou a propor soluções para tais desafios, melhorando o escalonamento dos SGWfC, e mais atualmente adaptando esses SGWfC para atenderem a execução de programas feitos para aceleradores GPU, traduzindo programas clássicos para linguagem GPGPU e por fim criando linguagens que podem ser compiladas para múltiplas plataformas.

No âmbito de aprimorar um SGWfC para este ser capaz de executar atividades programadas para GPU podemos citar os trabalhos envolvendo os esforços para executar uma pesquisa sobre dinâmica molecular no *kepler* [43], o SGWfC criado para resolver um problema de Métodos de Elementos Finitos [46], e o SGWfC criado para dar suporte a experimentos de bioinformática [57].

Iremos analisar estes trabalhos em três pontos principais, sendo eles a capacidade de execução em um ambiente GPGPU, a capacidade de captura de proveniência de execuções realizadas em ambientes GPGPU, e por fim a forma como o cientista deve informar ao SGWfC sua atividade que possui uma versão para GPU.

Estes trabalhos se assemelham bastante ao que foi apresentado nesta dissertação por além de se utilizar de um SGWfC, de alguma forma o adaptar para que este venha lidar com atividades que possuem uma versão para GPU, portanto iremos entrar em mais detalhes sobre cada um desses trabalhos nesta seção.

O trabalho de Método de Elementos Finitos [46] propõe um SGWfC com suporte a atividades GPU possuindo uma interface gráfica intuitiva para que o cientista consiga

montar seu *workflow* e executa-lo.

Porém nada é informado sobre a captura de proveniência retrospectiva das tarefas executadas em GPU, a falta desta informação prejudica a evolução da pesquisa, pois o cientista não saberá o ganho ou perda das execuções de deu experimento em diferentes configurações de máquinas. Além disto o SGWfC limita o cientista a utilizar a versão CPU ou GPU da atividade, o que impossibilita o cientista de utilizar as duas versões da sua atividade simultaneamente.

O trabalho sobre dinâmica molecular no Kepler foi bem sucedido não apenas em executar uma atividade em uma GPU, como também é capaz de executar em múltiplas GPUs, isto por utilizar uma versão do software AMBER que possibilita estas formas de execução.

Porém o controle da execução parte do usuário, ou seja, ele deve definir qual tarefas serão executadas em determinado ambiente. A proveniência prospectiva é capturada via programas paralelos que são invocados durante a execução do *workflow* principal, e nada é informado sobre a captura da proveniência retrospectiva.

Deste modo podemos observar que ele superou o apresentado nesta dissertação por ser capaz de utilizar múltiplos recursos para a execução de uma tarefa de seu *workflow*, porém a responsabilidade de definir este uso é do cientista.

O SGWfC voltado para problemas de bioinformática [57] propôs uma abordagem mais completa, incluindo a captura de proveniência retrospectiva das atividades, mesmo quando elas estão sendo executadas por GPUs.

Porém, como nas outras propostas ele não permite ao cientista fornecer mais de uma forma de execução de uma atividade, tornando do cientista a responsabilidade da escolha do ambiente no qual a atividade será executada.

Por outro lado, temos diversas pesquisas adaptando códigos utilizados em diversos experimentos para uma linguagem GPGPU. Temos como exemplo o próprio CMSA [9] utilizado nessa dissertação na modelagem do Sciphy, assim como os programas utilizados nos trabalhos citados acima.

O CSMA é uma adaptação do MSA para GPU, tal adaptação foi influenciada pelo MSA ser uma poderosa técnica para analise de sequencia e portanto já ser um programa bem difundido no meio acadêmico, portanto o CSMA é capaz de ser executado em ambientes GPU e CPU, ou seja, o cientista pode fornecer os dois modos de execução para o SGWfC e este decidir qual é o melhor para ser executado, e por tal motivo ele foi utilizado

nesta dissertação.

Capítulo 6

Conclusão e Trabalhos Futuros

Neste trabalho tivemos como objetivo adaptar o SGWfC SciCumulus a nova era dos *hardwares* aceleradores, a principio o tornando compatível com as GPUs CUDA da Nvidia.

Com isto agora podemos informar ao SciCumulus as versões que temos disponíveis de uma determinada atividade e este em tempo de execução escolher qual versão da atividade será executada, um passo importante para o cientista se preocupar apenas com a sua pesquisa, deixando a teoria computacional a cargo do SciCumulus.

Obtivemos sucesso ao executar o *workflow* SciPhy em um ambiente heterogêneo da Amazon, aproveitando o máximo possível das GPUs disponíveis.

Porém, este é apenas um primeiro passo, muitas melhorias podem ser feitas a partir deste arcabouço inicial, iremos agora discutir algumas destas melhorias.

6.1 Suporte a novos aceleradores

Atualmente o SciCumulus provê suporte para GPUs NVidia, que utilizam linguagem CUDA, um avanço importante é incluirmos o suporte a GPUs que utilizam a linguagem OpenCl, para darmos mais opções ao cientista.

Para esta inclusão ser feita é preciso levantar os dados relevantes ao OpenCl, e criar a lógica de detecção do acelerador, execução e captura de proveniência necessárias para este tipo de acelerador e dos dados levantados.

Outro ponto importante para melhorar a escalabilidade do SciCumulus será permitir que estes módulos de suporte a aceleradores possam ser *plug-ins* feitos por terceiros, e o cientista escolher nesta estante de *plug-ins* o que melhor lhe atende, seja pelo acelerador

que ele dá suporte ou pela qualidade da proveniência que ele captura.

6.2 Escalonamento das Tarefas

Temos neste aspecto dois grandes desafios, o primeiro é a escolha de qual atividade será executada em um determinado acelerador com base em toda a estrutura do *workflow*, a segunda é unir múltiplas *heads* para a execução de uma atividade que exige mais poder de processamento, como por exemplo múltiplas GPUs, ou uma união de CPU e GPUs.

Quanto ao primeiro ponto a grande questão é se vale a pena alocar uma atividade para uma determinada GPU se em breve pode vir uma atividade que só possa ser executada nesta GPU, ou seja, uma forma de prever e mitigar os possíveis gargalos durante a execução do *workflow*.

O segundo ponto se confunde um pouco com o primeiro, visto que para saber se vale a pena juntar múltiplos recursos para uma atividade devemos saber o impacto que isto causará para o restante da execução do *workflow*, e utilizar estes recursos em atividades que seriam gargalo para assim melhorar o desempenho geral da execução do *workflow*.

Tais melhorias não são triviais, visto que uma escolha errada de alocação de recursos pode criar diversos gargalos que anteriormente não existiriam, o que prova um trabalho desafiador prover tal tipo de escalonamento.

Referências

[1] Dicionário Brasileiro da Língua Portuguesa. `http://michaelis.uol.com.br/busca?id=5Bo7k`. Accessed: 2019-03-15.

[2] Abramson, D., Enticott, C., Altinas, I. Nimrod/K: Towards massively parallel dynamic Grid workflows.

[3] Anglano, C., Canonico, M. Scheduling Algorithms for Multiple Bag-of-Task Applications on Desktop Grids: a Knowledge-Free Approach.

[4] Anglano, C., Canonico, M., Guazzone, M., Botta, M., Rabellino, S., Arena, S., Girardi, G. Peer-to-Peer Desktop Grids in the Real World: The ShareGrid Project. Em *2008 Eighth IEEE International Symposium on Cluster Computing and the Grid (CCGRID)* (May 2008), p. 609–614.

[5] Beveridge, W. I. B. *The Art of Scientific Investigation.* 2004.

[6] Bose, R., Foster, I., Moreau, L. Report on the International Provenance and Annotation Workshop (IPAW'06) 3-5 May 2006, Chicago s.

[7] Buneman, P., Khanna, S., Tan, W.-C. Why and Where: A Characterization of Data Provenance.

[8] Callahan, S. P., Freire, J., Santos, E., Scheidegger, C. E., Silva, C. T., Vo, H. T. VisTrails: Visualization Meets Data Management. Em *Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data* (New York, NY, USA, 2006), SIGMOD '06, ACM, p. 745–747.

[9] Chen, X., Wang, C., Tang, S., Yu, C., Zou, Q. CMSA: a heterogeneous CPU/GPU computing system for multiple similar RNA/DNA sequence alignment. *BMC Bioinformatics 18*, 1 (Jun 2017), 315.

[10] Chen, X., Wang, C., Tang, S., YuEmail, C., Zou, Q. CMSA: a heterogeneous CPU/GPU computing system for multiple similar RNA/DNA sequence alignment.

[11] Cirne, W., Brasileiro, F., Andrade, N., Costa, L., Andrade, A., Novaes, R., Mowbray, M. Labs of the World, Unite!!!

[12] Cruz, S., Batista, V., Dávila, A., Silva, E., Tosta, F., Vilela, C., Campos, M., Cuadrat, R., Tschoeke, D., Mattoso, M. OrthoSearch: a scientific workflow approach to detect distant homologies on protozoans.

[13] Cruz, S., Campos, M., Mattoso, M. Towards a Taxonomy of Provenance in Scientific Workflow Management Systems.

[14] Davidson, S., Freire, J. Provenance and scientific workflows: challenges and opportunities.

[15] de Oliveira, D., Ocana, K., Ogasawara, E., Dias, J., Baiao, F., Mattoso, M. A Performance Evaluation of X-Ray Crystallography Scientific Workflow Using SciCumulus. Em *2011 IEEE 4th International Conference on Cloud Computing* (July 2011), p. 708–715.

[16] de Oliveira, D., Ocaña, K., Baião, F., Mattoso, M. A Provenance-based Adaptive Scheduling Heuristic for Parallel Scientific Workflows in Clouds. *Journal of Grid Computing 10* (09 2012).

[17] de Oliveira, D., Ogasawara, E., Baião, F., Mattoso, M. SciCumulus: A Lightweight Cloud Middleware to Explore Many Task Computing Paradigm in Scientific Workflows. Em *2010 IEEE 3rd International Conference on Cloud Computing* (July 2010), p. 378–385.

[18] de Oliveira, D., Viana, V., Ogasawara, E., Ocana, K., Mattoso, M. Dimensioning the Virtual Cluster for Parallel Scientific Workflows in Clouds. Em *Proceedings of the 4th ACM Workshop on Scientific Cloud Computing* (New York, NY, USA, 2013), Science Cloud '13, ACM, p. 5–12.

[19] Deelman, E., Gannon, D., Shields, M., Taylor, I. Workflows and e-Science: An overview of workflow system features and capabilities.

[20] Deelman, E., Vahi, K., Juve, G., Rynge, M., Callaghan, S., Maechling, P. J., Mayani, R., Chen, W., da Silva, R. F., Livny, M., Wenger, K. Pegasus, a workflow management system for science automation. *Future Generation Computer Systems 46* (2015), 17 – 35.

[21] Fahringer, T., Prodan, Duan, R., Nerieri, F., Podlipnig, S., Qin, J., Siddiqui, M., Truong, H.-L., Villazon, A., Wieczorek, M. ASKALON: A Grid Application Development and Computing Environment.

[22] Felsenstein, J. PHYLIP - Phylogeny Inference Package (Version 3.2). *Cladistics 5* (1989), 164–166.

[23] Freire, J., Koop, D., Santos, E., Silva, C. Provenance for Computational Tasks: A Survey.

[24] GEORGAKOPOULOS, D., HORNICK, M., SHETH, A. An Overview of Workflow Management: From Process Modeling to Workflow Automation Infrastructure.

[25] Gil, Y., Deelman, E., Ellisman, M., Fahringer, T., Fox, G., Gannon, D., Goble, C., Livny, M., Moreau, L., Myers, J. Examining the Challenges of Scientific Workflows.

[26] Gilbert, D. Sequence File Format Conversion with Command-Line Readseq. *Current Protocols in Bioinformatics 00*, 1 (2003), A.1E.1–A.1E.4.

[27] Goble, C. Position Statement: Musings on Provenance, Workflow and (Semantic Web) Annotations for Bioinformatics.

[28] Hung, C.-L., Lin, Y.-S., Lin, C.-Y., Chung, Y.-C., Chung, Y.-F. CUDA ClustalW. *Comput. Biol. Chem. 58*, C (outubro de 2015), 62–68.

[29] Jacob, J., Katz, D., Berriman, B., Good, J., Laity, A., Deelman, E., Kesselman, C., Singh, G., Su, M.-H., Prince, Thomas Williams, R. Montage: a grid portal and software toolkit for science-grade astronomical image mosaicking.

[30] Jr, H. G. G. *Scientific Method in Practice.* 2002.

[31] Keane, T. M., Creevey, C. J., Pentony, M. M., Naughton, T. J., Mclnerney, J. O. Assessment of methods for amino acid matrix selection and their use on empirical data shows that ad hoc assumptions for choice of matrix are not justified. *BMC Evolutionary Biology 6*, 1 (Mar 2006), 29.

[32] Langdon, W. Large-Scale Bioinformatics Data Mining with Parallel Genetic Programming on Graphics Processing Units.

[33] Ludäscher, B., Altintas, I. Scientific Workflow Management and the Kepler System.

[34] Mattoso, M., Werner, C., Travassos, G., Braganholo, V., Ogasawara, E., de Oliveira, D., Cruz, S., Martinho, W., Murta, L. Towards Supporting the Life Cycle of Large-scale Scientific Experiments. *International Journal of Business Process Integration and Management 5* (05 2010), 79–92.

[35] Moreau, L., Freire, J., Futrelle, J., McGrath, R., Myers, J., Paulson, P. The Open Provenance Model: An Overview.

[36] Moreau, L., Ludäscher, B., Altintas, I., Barga, R. S., Bowers, S., Callahan, S., Chin JR., G., Clifford, B., Cohen, S., Cohen-Boulakia, S., Davidson, S., Deelman, E., Digiampietri, L., Foster, I., Freire, J., Frew, J., Futrelle, J., Gibson, T., Gil, Y., Goble, C., Golbeck, J., Groth, P., Holland, D. A., Jiang, S., Kim, J., Koop, D., Krenek, A., McPhillips, T., Mehta, G., Miles, S., Metzger, D., Munroe, S., Myers, J., Plale, B., Podhorszki, N., Ratnakar, V., Santos, E., Scheidegger, C., Schuchardt, K., Seltzer, M., Simmhan, Y. L., Silva, C., Slaughter, P., Stephan, E., Stevens, R., Turi, D., Vo, H., Wilde, M., Zhao, J., Zhao, Y. Special Issue: The First Provenance Challenge. *Concurrency and Computation: Practice and Experience 20*, 5 (2008), 409–418.

[37] Ocaña, K., Oliveira, D., Ogasawara, E., Dávila, A., Lima, A., Mattoso, M. SciPhy: A Cloud-Based Workflow for Phylogenetic Analysis of Drug Targets in Protozoan Genomes.

[38] Ogasawara, E., Dias, J., Oliveira, D., Porto, F., Valduriez, P., Mattoso, M. An Algebraic Approach for Data-Centric Scientific Workflows.

[39] Oinn, T., Addis, M. Taverna: a tool for the composition and enactment of bioinformatics workflows.

[40] Oliveira, D., Ogasawara, E., Baião, F., Mattoso, M. SciCumulus: A Lightweight Cloud Middleware to Explore Many Task Computing Paradigm in Scientific Workflows.

[41] Oliveira, D., Ogasawara, E., Ocaña, K., Baião, F., Mattoso, M. An adaptive parallel execution strategy for cloud-based scientific workflows.

[42] Owens, J., Luebke, D., Govindaraju, N., Harris, M., Krüger, J., Lefohn, A., Purcell, T. A Survey of General-Purpose Computation on Graphics Hardware. 80–113.

[43] Purawat, S., Ieong, P. U., Malmstrom, R. D., Chan, G. J., Yeung, A. K., Walker, R. C., Altintas, I., Amaro, R. E. A Kepler Workflow Tool for Reproducible AMBER GPU Molecular Dynamics. *Biophysical Journal 112*, 12 (2017), 2469 – 2474.

[44] Richard D. Jarrard. Scientific Methods. `http://emotionalcompetency.com/sci/sm1.htm#4`, 2001. Online; accessed 07 Outubro 2018.

[45] Sahinidis, N. V., Vouzis, P. D. GPU-BLAST: using graphics processors to accelerate protein sequence alignment. *Bioinformatics 27*, 2 (11 2010), 182–188.

[46] Silva, V. Workflows Paramétricos para Aplicações do Método dos Elementos Finitos em Ambientes Paralelos Heterogêneos. Dissertação de Mestrado, Universidade Federal De Mato Grosso do Sul, 2013.

[47] Simmhan, Y., Plale, B., Gannon, D. A Survey of Data Provenance Techniques.

[48] Smanchat, S., Indrawan, M., Ling, S., Enticott, C., Abramson, D. Scheduling Multiple Parameter Sweep Workflow Instances on the Grid. Em *2009 Fifth IEEE International Conference on e-Science* (Dec 2009), p. 300–306.

[49] Sorde, S. W., Aggarwal, S. K., Song, J., Koh, M., See, S. Modeling and Verifying Non-DAG Workflows for Computational Grids. Em *2007 IEEE Congress on Services (Services 2007)* (July 2007), p. 237–243.

[50] Stamatakis, A. RAxML-VI-HPC: maximum likelihood-based phylogenetic analyses with thousands of taxa and mixed models. *Bioinformatics 22*, 21 (08 2006), 2688–2690.

[51] Taylor, I. J. *Workflows for e-Science: scientific workflows for grids*. 2007.

[52] Thain, D., Tannenbaum, T., Livny, M. Condor and the Grid. Em *Grid Computing: Making the Global Infrastructure a Reality*, F. B. G. F. T. Hey, Ed. 2004, cap. 11.

[53] Travassos, G., Barros, M. Contributions of In Virtuo and In Silico Experiments for the Future of Empirical Studies in Software Engineering Contributions of In Virtuo and In Silico Experiments for the Future of Empirical Studies in Software Engineering. 117.

[54] Travassos, G. H., de Oliveira Barros, M. Contributions of In Virtuo and In Silico Experiments for the Future of Empirical Studies in Software Engineering. *2nd Workshop on Empirical Software Engineering the Future of Empirical Studies in Software Engineering* (2003), 117–130.

[55] Vossen, G., Weske, M. The WASA Approach to Workflow Management for Scientific Applications.

[56] W3C. An Overview of the PROV Family of Documents. https://www.w3.org/TR/2013/NOTE-prov-overview-20130430/, 2013. Online; accessed 22 Janeiro 2019.

[57] Welivita, A., Perera, I., Meedeniya, D. An Interactive Workflow Generator to Support Bioinformatics Analysis Through GPU Acceleration. p. 457–462.

[58] Wieczorek, M., Prodan, R., Fahringer, T. Scheduling of scientific workflows in the ASKALON grid environment.

[59] Wozniak, J. M., Armstrong, T. G., Wilde, M., Katz, D. S., Lusk, E., Foster, I. T. Swift/T: Large-Scale Application Composition via Distributed-Memory Dataflow Processing. Em *2013 13th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing* (May 2013), p. 95–102.