UNIVERSIDADE FEDERAL FLUMINENSE

IGOR BARRETO RODRIGUES

Uma Abordagem Hierárquica para Escalonamento de Atividades de Workflows Científicos Executados em Nuvens de Computadores

Niterói

2016

UNIVERSIDADE FEDERAL FLUMINENSE

IGOR BARRETO RODRIGUES

Uma Abordagem Hierárquica para Escalonamento de Atividades de Workflows Científicos Executados em Nuvens de Computadores

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: Sistemas de Computação

Orientador:
Daniel Cardoso Moraes de Oliveira

Niterói

2016

Ficha Catalográfica elaborada pela Biblioteca da Escola de Engenharia e Instituto de Computação da UFF

R696 Rodrigues, Igor Barreto
Uma abordagem hierárquica para escalonamento de atividades de workflows científicos executados em nuvens de computadores / Igor Barreto Rodrigues. – Niterói, RJ : [s.n.], 2016.
48 f.

Dissertação (Mestrado em Computação) - Universidade Federal Fluminense, 2016.
Orientador: Daniel Cardoso Moraes de Oliveira.

1. Sistema de computador. 2. Computação nas nuvens. 3. Fluxo de trabalho. 4. Escalonamento de tarefa. I. Título.

CDD 004.2

IGOR BARRETO RODRIGUES

Uma Abordagem Hierárquica para Escalonamento de Atividades de *Workflows* Científicos Executados em Nuvens de Computadores

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: Sistemas de Computação

BANCA EXAMINADORA

Prof. Daniel Cardoso Moraes de Oliveira - Orientador, UFF
(Presidente)

Prof. Fábio André Machado Porto, LNCC

Profª. Vanessa Braganholo Murta, UFF

Niterói
2016

Aos meus pais e minha noiva.

Agradecimentos

Primeiramente gostaria de agradecer aos meus pais, Marco Antônio e Joselita Barreto, pelo apoio e incentivo que sempre me deram, priorizando sempre meus estudos.

Á minha noiva, Jamile Liberato, por todo apoio, incentivo e paciência durante toda essa caminhada, é sem dúvida a melhor companheira para estar ao lado.

Ao meu orientador, Daniel de Oliveira, pelos ensinamentos e por ter me acolhido e guiado nessa trajetória. Pela paciência de me explicar cada detalhe, acima de tudo ganhei um amigo.

Aos meus amigos do Instituto de Computação, Glaubos, Alberto, Gilberto, Heder, Rafael, Edcarlos, Marcos, Italo, Sávio, que direta ou indiretamente contribuíram para essa conquista.

Aos membros da banca Prof. Fabio Porto e Profa. Vanessa Braganholo.

Aos secretários do Instituto de Computação, principalmente a Teresa, que sempre foi muito prestativa e colaborou bastante, e ao IC/UFF pelo uso do laboratório.

Ao CNPq, pelo apoio financeiro.

Resumo

Os *Workflows* científicos são muito utilizados para modelar experimentos por cientistas que necessitam executar simulações que envolvem o processamento de um grande volume de dados. Para atingir tal objetivo, os cientistas fazem uso de ambientes de processamento de alto desempenho (PAD) e técnicas de paralelismo. Esse paralelismo envolve basicamente distribuir (escalonar) atividades dos *workflows* em ambientes distribuídos e heterogêneos. É bem conhecido que escalonar tarefas em ambiente de computação de alto desempenho é um problema NP-Completo. Muitos escalonamentos tradicionais buscam otimizar apenas o tempo total de execução, porém em cenários mais reais existem vários fatores a serem considerados, como por exemplo a confiabilidade, o consumo de energia e o custo financeiro. Devido a isto, algumas abordagens mais recentes já levam em consideração tais objetivos. No entanto muitas delas utilizam funções objetivo com pesos que podem se tornar um problema, pois a medida que aumenta o número de objetivos a diferença dos pesos se torna cada vez menor. Essa dissertação propõe uma abordagem de escalonamento baseada em quatro objetivos para ambiente de nuvens, são eles: o tempo de execução, o custo monetário, a confiabilidade e o custo energético, em que é construída uma lista de escalonamentos como solução, onde todos os objetivos são levados em consideração em uma determinada ordem estabelecida pelo cientista. Como forma de instanciar essa solução, essa proposta foi implementada estendendo o SciCumulus. Com o objetivo de validar a proposta, foram executados testes em um *workflow* sintético baseado no *CloudSim* e também em um *workflow* real na nuvem da Amazon EC2 executando o SciPhy com o SciCumulus, um sistema de gerência de *workflow* científico (SGWfC). Os resultados obtidos apresentaram melhoras no fator escolhido para otimização, fazendo com que as várias necessidades do usuário sejam atendidas, tendo portanto, um SGWfC mais eficiente e um cenário mais real.

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

Abstract

Scientific workflows are widely used by scientists to process a large volume of data with high performance computing (PAD) in heterogeneous environments. Scheduling tasks in a high-performance computing environment is an NP-Hard problem. Traditional scheduling research usually targets only optimizing makespan, altough in real scenarios there are several factors to be considered, for instance reliability, energy consumption and financial cost. Due to this, some recent approaches already take into account other objectives, but most use functions with weights that end up optimizing an objective, and the measure that increases the number of objectives the difference of weights between them are dwindling. This dissertation proposes an scheduling approach with four goals for cloud environment, they are: makespan, economic cost, reliability and energy consumption, so is built a scheduling list as solution, where all objectives are taken into account. As a way to instantiate a solution, this proposal was implemented extending SciCumulus. Aiming to validate the proposal we have performed some tests in a sintetic workflow based on CloudSim and also in a real workflow in the cloud Amazon EC2 using SciPhy with SciCumulus, a scientific workflow management system. The results demonstrate improvement in the cost of all objectives, causing the various user needs been attended, and therefore, a scientific workflow management system more efficient and a more realistic scenario.

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

Lista de Figuras

1.1 Cenários representando possíveis escalonamentos 3

2.1 Ciclo de vida de um experimento científico adaptado de Mattoso *et al.* (2010) 10

2.2 Grafo G representando o fluxo de dados de um *workflow* 11

3.1 Exemplo de escalonamento hierárquico 26

3.2 Exemplo de restrição de escalonamento hierárquico 26

4.1 Gráfico Tempo de Execução *Workflow* Sintético 29

4.2 Gráfico Custo Financeiro *Workflow* Sintético 30

4.3 Gráfico Custo Energético *Workflow* Sintético 31

4.4 Gráfico Tempo de Execução *Workflow* Real 32

4.5 Gráfico Confiabilidade *Workflow* Real 33

4.6 Gráfico Custo Financeiro *Workflow* Real 34

4.7 Gráfico Consumo Energético *Workflow* Real 35

Lista de Tabelas

1.1 Exemplo 1 - Distribuição de pesos entre objetivos 4

1.2 Exemplo 2 - Distribuição de pesos entre objetivos 4

4.1 Lista de Máquinas *Cluster* Sintético 28

4.2 Lista de Máquinas *Cluster* Real 32

5.1 Comparativo de abordagens 38

Lista de Abreviaturas e Siglas

PAD	:	Processamento de Alto Desempenho
SGWfC	:	Sistema de Gerência de *Workflow* Científico
XML	:	*Extensible Markup Language*
SBC	:	Sociedade Brasileira de Computação
QoS	:	*Quality of Service*

Sumário

1 Introdução **1**

1.1 Apresentação do Problema 2

1.2 Hipótese 6

1.3 Objetivos 6

1.4 Organização da Dissertação 7

2 Fundamentação Teórica **8**

2.1 Experimentos Científicos 8

2.2 Workflows Científicos 11

2.3 Proveniência de Dados 11

2.4 Escalonamento de *workflows* em nuvens 13

2.5 Formalização do Cenário 14

2.5.1 Formalização do ambiente de nuvem 14

2.5.2 Tempo de execução 15

2.5.3 Transferência de dados 16

2.5.4 Confiabilidade 17

2.5.5 Custo Monetário 18

2.6 Algoritmo de Escalonamento 19

3 Escalonamento hierárquico multiobjetivo de workflows em nuvens de computadores **21**

3.1 Função Objetivo 21

3.1.1 Consumo Energético 22
3.2 Abordagem proposta 23

4 Avaliação Experimental **27**
4.1 Metodologia do experimento 27
4.2 *Workflow* Sintético 28
4.3 SciPhy 30
4.4 *Workflow* Real 31

5 Trabalhos Relacionados **36**

6 Conclusão e Trabalhos Futuros **39**

Referências Bibliográficas **41**

Capítulo 1

Introdução

Os experimentos científicos computacionais surgem com a necessidade de se estudar, reproduzir experimentos e provar empiricamente fenômenos por meio de simulações (Mattoso *et al.*, 2009). Certos experimentos envolvem uma grande quantidade de aplicações encadeadas que produzem um grande volume de dados e necessitam de um grande poder computacional para processá-las, além de um mecanismo para gerenciar as execuções de tais encadeamentos. Além disso, cada experimento científico é concebido a fim de confirmar ou refutar uma hipótese científica. Para isso, o cientista pode necessitar explorar muitas variações de parâmetros (Ogasawara *et al.*, 2009). Devido a isto, buscando reduzir custos de tempo e valor monetário, tradicionalmente usamos ambientes de processamento de alto desempenho (PAD) aliados à técnicas de paralelismo (Oliveira, 2012). Aplicações paralelas com restrições de precedência entre suas etapas, conhecidas como *workflows* científicos são um paradigma popular usado por cientistas para a modelagem de simulações computacionais complexas (Mattoso *et al.*, 2010). A maioria destas simulações tem que entregar resultados tão rápido quanto possível e, portanto, exigem computação paralela ou distribuída para diminuir o seu tempo de execução (Durillo *et al.*, 2014). Devido à complexidade da gerência do fluxo de dados, os *workflows* facilitam os inúmeros experimentos feitos pelos cientistas.

Dessa forma, as simulações podem ser modeladas como um conjunto de atividades e um fluxo de dados entre as mesmas. Cada atividade do *workflow* pode ser um programa de computador ou um serviço *Web* que processa um conjunto de dados de entrada e produz um conjunto de dados de saída (Oliveira *et al.*, 2010). Os *workflows* são controlados por sistemas complexos chamados de Sistemas de Gerência de *Workflows* Científicos (*i.e.* SGWfC). Esses sistemas são capazes de executar o *workflow* em ambientes heterogêneos, como por exemplo, a nuvem de computadores (Vaquero *et al.*, 2008). Se seguirmos a

abordagem algébrica para representação de *workflows* proposta por Ogasawara *et al.* (2011), as atividades do *workflow* são divididas em ativações (que são tarefas menores) que são a menor unidade de processamento possível. Os SGWfC paralelos baseados nese modelo algébrico distribuem as ativações para os recursos computacionais, respeitando a dependência entre elas (Sousa, 2014).

A nuvem computacional fornece um ambiente totalmente favorável para estes tipos de aplicações, pois além da alta disponibilidade, é fornecida a ideia de elasticidade, *i.e.* a capacidade de aumentar e/ou diminuir o tamanho do *cluster* de máquinas virtuais criado para processar o *workflow* de acordo com a necessidade, e o usuário paga apenas pelos recursos que consumir *(pay-per-use)* (Vaquero *et al.*, 2008). Algumas empresas destacam-se neste cenário, como por exemplo a Amazon EC2[1], Google Cloud Platform[2], Locaweb[3], Microsoft Azure[4]. Neste trabalho foi utilizado a plataforma da Amazon EC2.

1.1 Apresentação do Problema

O escalonamento de tarefas é um conhecido problema NP-difícil (não polinomial), mesmo em suas formas mais simples (Ullman, 1975). Existem vários requisitos para serem abordados quando se vai escalonar uma tarefa para executar em determinada máquina, como por exemplo, questões de desempenho, falhas, taxas de transferência, que podem variar de acordo com a tarefa e o objetivo a ser otimizado. Agregando a este cenário, escalonar tarefas em uma nuvem de computadores se torna ainda mais complicado, pois ela pode apresentar variações ainda maiores no desempenho, nas taxas de transferências, e ainda há outros requisitos a serem considerados como por exemplo o custo que vai ser gerado pelos provedores de serviço, tanto pelo aluguel das máquinas, como pelo volume de dados transferidos durante a execução do *workflow*.

Uma das heurísticas muito comuns em escalonamento de *workflows* é a atribuição de pesos aos objetivos que deseja-se otimizar. Embora seja uma abordagem eficaz, apresentam-se dois problemas:

(i) Ao se executar um *workflow* pela primeira vez, o usuário não tem ideia alguma de como ele irá se comportar baseado nos pesos de entrada, somente após várias execuções e ajustes é que ele conseguirá obter o resultado que realmente deseja ou o mais próximo

[1]https://aws.amazon.com/pt/ec2/
[2]https://cloud.google.com/
[3]www.locaweb.com.br/cloud/
[4]https://azure.microsoft.com/pt-br/

disso. A Figura 1.1 mostra 3 cenários com escalonamentos diferentes. Suponhamos que um usuário não tenha muita experiência em executar determinado *workflow* na nuvem. Agora supomos que ele crie um *cluster* com quatro máquinas (duas máquinas robustas, com maior poder de processamento e duas máquinas lentas, com menor poder de processamento). Após a criação do *cluster*, faz uma distribuição de pesos e executa o *workflow*. O cenário A apresenta o cenário ideal, as máquinas robustas teriam prioridade sobre as máquinas lentas. No entanto, o cenário C que é totalmente o oposto do cenário A (desejado pelo usuário) foi o que ocorreu durante a execução, e as máquinas lentas tiveram prioridade total em relação as máquinas robustas.

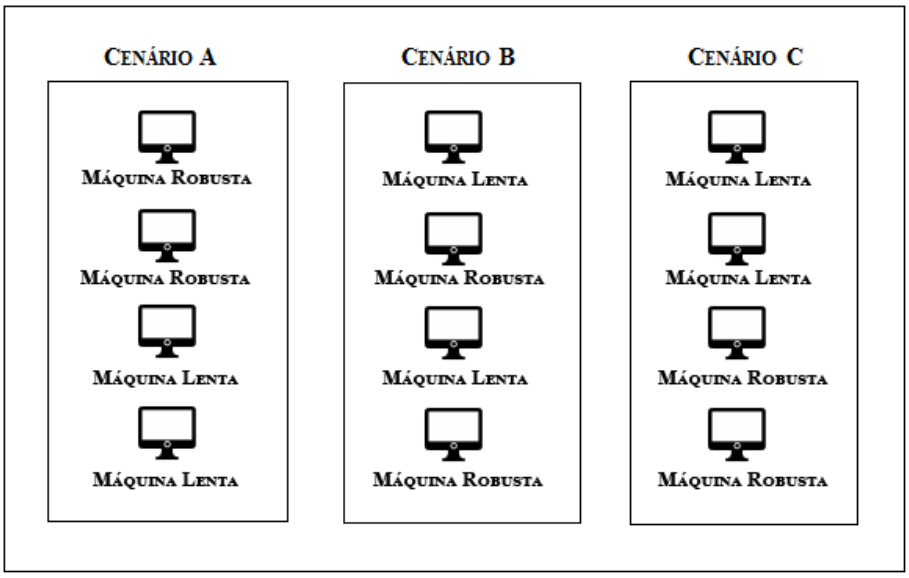

Figura 1.1: Cenários representando possíveis escalonamentos

O exemplo citado é o pior dos casos, vamos supor agora que o usuário ao ver o comportamento anterior tentou ajustar os pesos e executou o *workflow* novamente, porém ainda aconteceu o cenário B, em que as máquinas lentas e robustas se misturaram e a que teve maior prioridade ainda foi uma máquina lenta, apesar de ter melhorado ainda é necessário alterar os pesos e fazer a execução novamente para atingir o cenário desejado. Portanto, somente após várias execuções o usuário irá aprendendo a fazer os ajustes necessários dos pesos e obter o comportamento que realmente deseja (cenário ideal).

(ii) a medida que adicionamos mais objetivos ao modelo, mais difícil se torna a função de otimização, uma vez que a diferença de peso entre os critérios ficará pequena, se tornando quase irrelevante e fazendo com que o escalonamento perca o foco. Vamos supor que o usuário tenha três objetivos para otimizar, e ele tem que fazer uma distribuição de pesos entre eles, o somatório dos pesos tem que ser igual a 1. O usuário então fez um balanceamento como mostra na Tabela 1.1. Podemos perceber claramente que ele

está priorizando o tempo de execução, em seguida o custo financeiro e por último a confiabilidade.

Objetivo	Tempo de execução	Custo financeiro	Confiabilidade
Peso (entre 0 e 1)	0.5	0.3	0.2

Tabela 1.1: Exemplo 1 - Distribuição de pesos entre objetivos

Agora vamos supor que tenhamos adicionado o objetivo consumo de energia à função de custo. A Tabela 1.2 mostra a distribuição de pesos, ainda que seja possível distribuir de uma forma em que o objetivo tempo de execução fique maior que os demais objetivos, a medida que adicionarmos mais objetivos, mais complicado ficará fazer essa distribuição e os pesos vão assumindo valores cada vez menores. Para critérios didáticos, analisemos o que ocorreria nesse exemplo se o usuário fizesse essa distribuição de pesos como mostra a Tabela 1.2.

Objetivo	Tempo de execução	Custo financeiro	Confiabilidade	Consumo de Energia	...
Peso (entre 0 e 1)	0.3	0.3	0.2	0.2	...

Tabela 1.2: Exemplo 2 - Distribuição de pesos entre objetivos

Para calcular o custo para se executar uma tarefa em uma determinada máquina x usaremos a seguinte Equação:

$$a * (bx) + c * (dx) + e * (fx) + g * (hx) \tag{1}$$

Das variáveis da Equação 1 temos:

- a: Peso tempo de execução
- bx: Custo do tempo de execução na máquina x
- c: Peso custo financeiro
- dx: Preço de execução na máquina x
- e: Peso confiabilidade
- fx: Confiabilidade da máquina x
- g: Peso consumo de energia
- hx: Preço consumo de energia da máquina x

Conforme a distribuição feita na Tabela 1.2 e utilizando a Equação 1, primeiro vamos calcular o custo para a máquina mais lenta presente no *cluster* do exemplo citado em (i), novamente, para fins didáticos, vamos considerar em que 1 representa o pior custo de um objetivo e 10 o melhor custo, dessa forma, substituindo os valores teríamos:

$$0.3 * (1) + 0.3 * (10) + 0.2 * (1) + 0.2 * (10) = 5.5 \quad (2)$$

Na Equação 2 a variável bx assumiu o valor 1 pois consideramos que a máquina mais lenta do *cluster* tem o pior tempo de execução. A variável dx assumiu o valor 10 pois logo tem o melhor custo (mais barato), fx tem a pior confiabilidade, logo 1 e hx o melhor consumo de energia (pouco consumo), portanto 10. Dessa forma, teríamos o custo dessa máquina em 5.5.

Agora vamos refazer esse cálculo para a máquina robusta:

$$0.3 * (10) + 0.3 * (1) + 0.2 * (10) + 0.2 * (1) = 5.5 \quad (3)$$

Conforme vimos na Equação 3 o resultado obtido foi o mesmo da Equação 2. Logo, a máquina lenta e a robusta teriam o mesmo valor da função de custo para o escalonador e consequentemente ambos escalonamentos teriam a mesma prioridade mesmo que isso não representasse a realidade. Este erro acontece devido aos valores dos pesos ficarem muito próximos, e esse problema aumenta cada vez que adicionamos mais objetivos a função de otimização. Portanto, uma abordagem que utiliza pesos, conforme estes cenários explicados pode não ter um bom desempenho.

O SciCumulus (Oliveira, 2012) é um SGWfC adaptativo em relação ao estado dos recursos computacionais na nuvem e foi escolhido para instanciar as soluções propostas nessa dissertação. Ele é capaz de verificar a capacidade computacional e ajustar dinamicamente a distribuição de ativações, se necessário até redimensionar o tamanho do *cluster*, afim de alcançar um melhor desempenho. Seu escalonador leva em consideração três objetivos: o tempo de execução, o custo monetário e a confiabilidade. Esse SGWfC possui em sua arquitetura três módulos: O banco de dados de proveniência, o SciCumulus *Starter* e o Scicumulus *Core*, este último foi o foco nesta dissertação, embora todos eles tenham sido extensivamente usados para efetuar a pilha de experimentos. Mais detalhes sobre o funcionamento de cada módulo do SciCumulus podem ser visto em (Oliveira *et al.*, 2010; Oliveira, 2012).

Nessa dissertação, ao invés de utilizarmos uma função multi-objetivo com pesos propo-

mos o uso de várias funções objetivo de forma hierárquica para otimizar múltiplos critérios do usuário, fazendo um balanceamento entre as restrições de custo, tempo de execução, tolerância a falhas e o custo energético. Essa nova abordagem não utiliza mais pesos aos objetivos, e o cientista escolhe a ordem de prioridade dos objetivos, montando portanto uma hierarquia, e o resultado do escalonamento não é apenas uma solução, e sim um conjunto de escalonamentos que atendam aos objetivos do usuário e fazendo o *workflow* escolher os melhores custos de cada objetivo, deixando a execução mais adaptável.

Os testes experimentais foram realizados comparando-se a versão anterior do SciCumulus, onde a função objetivo utiliza pesos. Os resultados apresentaram uma melhora considerável de desempenho no escalonador do SGWfC, maximizando os ganhos definidos pela ordem de prioridade do usuário. Foram feitos testes com um *workflow* sintético e com um *workflow* real da bioinformática, alternando o escalonador nestes dois tipos de abordagens, e novamente os resultados se mostraram superiores.

1.2 Hipótese

Diante do cenário exposto na seção 1.1, a hipótese dessa dissertação é que se for utilizado uma abordagem hierárquica, substituindo assim à atribuição de pesos por uma lista ordenada de objetivos por prioridade, será montado uma lista de escalonamentos para cada objetivo, portanto o escalonador terá mais opções então conseguirá ter melhor desempenho.

1.3 Objetivos

Os objetivos desse trabalho são: Modificar o escalonador para construir uma lista de escalonamentos de forma hierárquica, possibilitando serem adicionados quantos objetivos forem necessários futuramente. Adicionar uma métrica de consumo de energia. Portanto, tornar o *workflow* com cenário mais real. Para isso, será instanciado essas soluções no SGWfC SciCumulus, como forma de consolidar a solução proposta nessa dissertação.

1.4 Organização da Dissertação

O restante desta dissertação está estruturado da seguinte maneira. O Capítulo 2 trata a respeito do referencial teórico necessário para melhor entendimento do trabalho desen-

volvido. O Capítulo 3 refere-se a abordagem proposta, detalhando como o trabalho foi desenvolvido. O Capítulo 4 apresenta a avaliação experimental. Em seguida, o Capítulo 5 aborda os trabalhos relacionados. Finalmente, o Capítulo 6 conclui a dissertação e discute sobre trabalhos futuros.

Capítulo 2

Fundamentação Teórica

Este capítulo tem como objetivo fornecer a fundamentação teórica necessária para se entender melhor sobre experimentos científicos em larga escala, bem como *workflows* científicos e seu funcionamento.

2.1 Experimentos Científicos

Segundo Mattoso *et al.* (2009), um experimento é uma das formas utilizadas pelos cientistas para apoiar a formulação de novas teorias. Soanes e Stevenson (2003), definem como "um teste executado sob condições controladas, que é realizado para demonstrar uma verdade conhecida, examinar a validade de uma hipótese, ou determinar a eficácia de algo previamente não explorado". Existem diversas formas de experimentos científicos, incluindo aqueles baseados no método tradicional (*in-vivo, in-vitro*). Novas modalidades vêm ganhando espaço na produção de conhecimento, onde as análises do experimento são feitas por simulações computacionais (*in-virtuo, in-silico*) (Travassos e Barros, 2003).

Nestas novas modalidades, as simulações são feitas por procedimentos computacionais, que têm sido grandes aliados de físicos, químicos, geólogos e cientistas de um modo geral (Mattoso *et al.*, 2009). Tanto que um termo foi criado para isto, *e-science*, que caracteriza-se pelo apoio ao cientista para o desenvolvimento de ciência em larga escala utilizando infraestrutura computacional correspondente (Mattoso *et al.*, 2008). Estes experimentos são simulados nas mais diversas áreas, como por exemplo, a prospecção de petróleo em águas profundas (Martinho *et al.*, 2009; Ogasawara *et al.*, 2011), a agricultura (Fileto *et al.*, 2003; Cho *et al.*, 2010), a astronomia (Berriman *et al.*, 2007; Freudling *et al.*, 2013), as análises filogenéticas (Ocaña *et al.*, 2011), o monitoramento aquático (Pereira e Ebecken, 2011), o processamento de sequências biológicas (Lemos *et al.*, 2004; Vicario

et al., 2012), etc.

Porém, para conseguir bons resultados, um número muito grande de execuções com as mais variadas definições de parâmetros e entradas deve ser utilizada, gerando um grande volume de dados, além de levar um longo período de tempo, o que acaba atrapalhando o progresso dos projetos científicos. Considerando-se os esforços para atingir elevados ganhos de produtividade e qualidade para se conseguir um diferencial competitivo (Mattoso *et al.*, 2009), faz-se necessário utilizar ferramentas que usem processamento distribuído e em larga escala. Este tema foi destacado como um dos grandes desafios pelo documento da Sociedade Brasileira de Computação (SBC) (SBC, 2006), com intuito de fornecer o apoio computacional ao desenvolvimento de ciência em larga escala. Neste sentido, foram desenvolvidos SGWfC, que serão melhor abordados na seção 2.2, e são uma alternativa atraente para a tarefa não trivial de controlar um grande volume de dados e contornar falhas de execuções.

De modo a melhor introduzir SGWfC no processo científico, o ciclo de vida de um experimento científico é definido por Mattoso *et al.* (2010) e mostrado na Figura 2.1. Esse ciclo é composto de três fases principais: composição, execução e análise. As fases principais e seus sub-ciclos independentes são percorridas inúmeras vezes durante o curso de um experimento e os sub-ciclos podem ser percorridos em momentos distintos.

A fase de composição é a responsável principalmente por instituir a sequência lógica de atividades, ou seja, define o fluxo do *workflow*. Também define a estrutura e a criação de todo o experimento, assim como os tipos de dados de entrada, quais parâmetros devem ser fornecidos e os tipos de dados de saída que são gerados. Suas duas sub-fases são a concepção e o reuso (mostrado na parte de cima da figura 2.1), enquanto a concepção é responsável pela criação do experimento, o reuso como o próprio nome já diz, é responsável por recuperar um experimento existente e adaptá-lo.

A fase de execução, mostrada na parte direita da Figura 2.1, materializa o experimento, tornando executável por um SGWfC a especificação do *workflow*. Logo, neste ciclo devem ser definidos exatamente os dados de entrada e os parâmetros que serão fornecidos ao SGWfC para executar o experimento. A primeira sub-fase é a distribuição, que devido a vários motivos, como o desempenho, está relacionada à necessidade de execução em ambientes com alto poder de processamento. E a segunda sub-fase é o monitoramento, devido a necessidade de checar o estado da execução da ferramenta, pois essa fase pode levar um longo período de tempo.

A Análise, é a terceira e última fase, que aparece no canto esquerdo da Figura 2.1.

Nesta fase é feito o estudo dos dados gerados nas duas fases anteriores. Ela é composta pelas sub-fases de visualização e consulta. Essa fase contém a chave de decisão do experimento, pois se o resultado estiver correto, como esperado pelo cientista, o experimento provavelmente deverá ser executado novamente para validar efetivamente os resultados (Mattoso *et al.*, 2010). Mas caso o resultado não seja conforme esperado, ou seja a hipótese foi refutada, logo ocasionará na criação/reformulação de uma outra hipótese ou ajustes de parâmetros. Se houver necessidade de várias execuções para validar uma hipótese, todas elas, independente dos dados de entradas e parâmetros, devem estar ligados ao mesmo experimento científico.

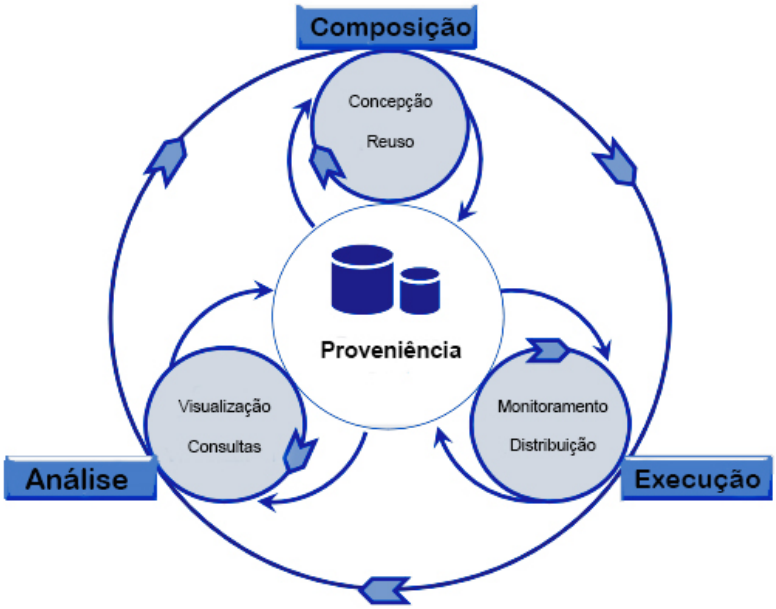

Figura 2.1: Ciclo de vida de um experimento científico adaptado de Mattoso *et al.* (2010)

2.2 Workflows Científicos

Os experimentos científicos em larga escala são compostos por um fluxo de dados que formam uma cadeia de atividades que são representadas como *workflows* científicos (Deelman *et al.*, 2009). Como vimos na seção anterior, os *workflows* científicos têm sido adotados nas mais diversas áreas, como biologia, agricultura, agronomia, astrologia, etc. E embora o significado da palavra *workflow* possa variar de acordo com a área, o termo Sistema de Gerência de *Workflow* Científico (SGWfC) guarda um significado comum nessas áreas: descrição de procedimentos científicos complexos, automatização dos processos de derivação de dados, processamento de alto desempenho para melhorar o desempenho, gerenciamento de proveniência e consulta (Zhao *et al.*, 2008).

Um *workflow* pode ser representado como um grafo acíclico dirigido *G (A, Dep)*, onde os nós representam as atividades a serem executadas e as arestas representam as dependências entre elas *(Dep)*. Como mostrado na Figura 2.2, que representa o fluxo de dados de um *workflow* científico, o grafo G, é composto por um conjunto de atividades $A = \{a_1, a_2, ..., a_n\}$, onde cada atividade tem seu conjunto de entrada de dados, representado pelo conjunto $I = \{i_1, i_2, ..., i_n\}$. Da mesma forma, cada atividade a_i, produzirá uma saída pertencente ao conjunto $O = \{o_1, o_2, ..., o_n\}$, em um determinado tempo de execução, em que $i_n \neq o_n$. Logo, a dependência entre duas atividades é dada por:

$$Dep(x_i, x_j, vd) \leftrightarrow \exists\ o_k \in Entrada(x_i)\ \mid\ o_k \in O(x_j)$$

onde vd é o volume de dados transferidos entre duas atividades x_i e x_j.

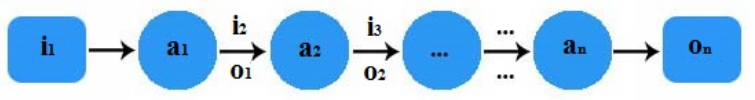

Figura 2.2: Grafo G representando o fluxo de dados de um *workflow*

2.3 Proveniência de Dados

O termo proveniência de dados se refere ao histórico de criação de dados de um objeto. A importância da proveniência de dados não pode ser subestimada, pois ela nos possibilita reproduzir os resultados. A reprodutibilidade é um componente crítico do método científico. A proveniência contém informações sobre o processo e os dados usados para derivar aquela informação, fornece uma documentação importante que é a chave para preservar os dados, determinar a qualidade deles, a autoria e reprodutibilidade, bem como a validação dos resultados (Davidson e Freire, 2008). Goble *et al.* (2003) sumarizam diversas características relevantes colhidas da análise de proveniência:

- Garantia de qualidade dos dados
  - Estimar a qualidade e confiabilidade com bases na origem dos dados e suas transformações registradas.
- Verificação de atribuição
  - Manter controle sobre as informações do dono do experimento e seus dados.
- Auditoria dos caminhos

  - Traçar rotas, determinando a utilização de recursos e detectando erros na geração dos dados.

- Informacional

  - Realizar consultas baseadas nos descritores de origem para a descoberta de dados. Provê o contexto necessário para melhor entendimento do mesmo.

Existem duas formas distintas de proveniência (Clifford *et al.*, 2008; Cruz *et al.*, 2009; Davidson e Freire, 2008; Freire *et al.*, 2008; Simmhan *et al.*, 2005): Prospectiva e Retrospectiva. A prospectiva contém a informação de um *workflow*, correspondendo aos passos que precisam ser seguidos (ou uma receita) para gerar os dados ou classes de um determinado produto.

A retrospectiva captura os passos que foram executados bem como a informação sobre o ambiente de execução usado para derivar a especificação de um produto, ou seja, um *"log"* detalhado da execução da tarefa computacional, com o tempo de execução, arquivos produzidos, erros, informações de desempenhos, entre outros (Oliveira, 2012).

Uma peça importante é a causalidade: a relação de dependência entre os dados e os processos que o geraram (Davidson e Freire, 2008). A proveniência pode ser coletada de diferentes formas e em múltiplos níveis de detalhes (Simmhan *et al.*, 2005), pois sua utilidade em certo domínio está vinculada ao nível de granularidade que é coletada, definindo o nível de detalhe de um produto de dados. A granularidade pode ser classificada como grossa *(coarse-grained)* ou fina *(fine-grained)*.

Segundo Buneman e Tan (2007) a granularidade fina refere-se ao registro da linhagem de um item específico de dado e não pode ser dividido em pedaços menores. A granularidade grossa, por sua vez, está relacionada ao registro do histórico completo de um conjunto de dados, gerado pelo processamento de um *workflow*, podendo conter dados não somente gerados no processamento, mas também de dispositivos e ambientes externos, como sensores, câmeras, interação humana (Cruz *et al.*, 2009).

2.4 Escalonamento de *workflows* em nuvens

Conforme mencionado anteriormente, o problema de escalonamento de *workflows* é bem desafiador, e no cenário de nuvem se torna ainda mais complexo. Nesse sentido, várias heurísticas têm sido propostas para ambientes heterogêneos (Assayad *et al.*, 2004;

Boeres *et al.*, 2011; Dongarra *et al.*, 2007; Hakem e Butelle, 2007; Qin e Jiang, 2005). No entanto, apesar de gerarem soluções ótimas, seu escalonamento é estático, ou seja, todo o plano de escalonamento é gerado **antes** do início da execução do *workflow*, e não consegue se adaptar a possíveis problemas que ocorrem durante a execução, como por exemplo uma máquina virtual ser reiniciada. Com isso todo o escalonamento feito pode se tornar inválido, portanto não adequado para o ambiente de nuvem de computadores.

Diferentemente de *grids*, para os quais é necessário um alto investimento inicial e portanto menos acessível a um público maior, a nuvem fornece um ambiente elástico (Vaquero *et al.*, 2008), com os recursos podendo ser solicitados sob demanda, de acordo com o necessário e pagando apenas pelo recurso que consumir *(pay-per-use)*. Dessa forma, o mais adequado seria um escalonamento adaptativo, fornecendo um melhor gerenciamento sob demanda. Nesse quesito, já existem SGWfCs como o Pegasus (Deelman *et al.*, 2007) e o Swift/T (Zhao *et al.*, 2007) que fornecem uma execução em nuvens, porém deixam de lado importantes requisitos desse ambiente, como por exemplo, tratar variações de desempenho.

Recentemente, vários sistemas de gerência de *workflows* científicos têm sido propostos para otimizar as questões levantadas nesta seção. Eles tratam o escalonamento de forma multiobjetiva, levando em consideração vários requisitos que atendam às necessidades do usuário e que estão adaptados ao cenário da nuvem (Oliveira, 2012; Fard *et al.*, 2012, 2014; Durillo *et al.*, 2014; Duan *et al.*, 2014; Zhang *et al.*, 2014).

Estes trabalhos visam em tempo de execução fornecer qualidade de serviço *(QoS) i.e.* redução de custos, consumo de energia, confiabilidade, aumento de desempenho, taxas de transferências, entre outros. No entanto, quanto mais objetivos são adicionados ao escalonamento baseado em uma função multiobjetivo, menor se torna o peso distribuído entre elas, de tal forma que chega a um ponto em que todos os objetivos se tornam quase que equivalentes, necessitando portanto de uma abordagem diferente, hierarquizada, que é o objetivo dessa dissertação e será melhor abordada no capítulo 3.

2.5 Formalização do Cenário

Antes de explicar melhor o funcionamento da abordagem proposta, precisamos formalizar o ambiente de nuvem e entender como é calculado o custo de um escalonamento usando uma função objetivo. Para isso apresentamos o formalismo proposto por Oliveira (2012) nessa seção. O processo de validação de uma hipótese através da execução de um

workflow pode ser longo. Apesar de, geralmente, o cientista ter pressa para obter os resultados, o que o levaria a executar o experimento com máquinas mais robustas, nem sempre terá fundos suficientes para isso. Existem métricas que podem ser escolhidas na hora da execução, essas funções definem o modelo de custo do SciCumulus (implementação escolhida para instanciar a solução proposta) e já foram bem consolidadas em (Oliveira *et al.*, 2010, 2012a,b; Oliveira, 2012). No entanto, essa dissertação estende a solução adicionando um objetivo a ferramenta, ele será tratado no capítulo 3. As próximas seções introduzem o formalismo das funções.

2.5.1 Formalização do ambiente de nuvem

O formalismo aqui exposto bem como o modelo de custo de cada função objetivo é baseado nas ideias de Oliveira (2012). Para formalizar as variáveis utilizadas no ambiente de nuvem, iremos considerar $VM = \{VM_0, VM_1, ..., VM_{n-1}\}$ como o conjunto de n máquinas virtuais criadas na nuvem e disponíveis para execução e gerência do SGWfC. E para cada máquina virtual, existe um valor associado a ela chamado de índice de retardo de computação *(csi)* (Oliveira, 2012). Nesta dissertação o valor de *csi* é obtido com base nos dados de proveniência das execuções anteriores do *workflow*, ou seja, é calculada uma média baseada nos tempos de execuções passadas de tarefas executadas por determinada máquina e que estão registados na base de dados. Este índice pode variar de 0 até 1, desta forma, uma máquina com desempenho máximo será atribuído o valor 1 e as restantes com valores relativos *(i.e. 0,72)*. Este índice também pode ser obtido através da função *csi*(VM_j). Uma tarefa a ser executada é modelada como uma *cloud activity* (ac_i) em Oliveira (2012) e será utilizado aqui da mesma forma.

2.5.2 Tempo de execução

Utilizando portanto das definições feitas em 2.5.1, considere $P(ac_i, VM_j)$ como sendo o tempo de execução previsto de uma tarefa ac_i em uma máquina virtual VM_j na nuvem. Da mesma forma, $P(ac_i, VM_j) = time(ac_i) \times csi(VM_j)$, em que $time(ac_i)$ é uma função que retorna o tempo médio desnormalizado da execução da tarefa, ou seja, faz a média com o tempo das execuções anteriores dividido pelo valor de *csi* das máquinas onde as tarefas foram executadas.

Agora considere também $\varphi(W, VM)$ como sendo o escalonamento total das tarefas pertencentes a um grupo AC do *workflow* W em VM. Dado um *workflow* W que

possua uma série de atividades $A = \{a_1, a_2, ..., a_n\}$ e um conjunto de tarefas $AC = \{ac_1, ac_2, ..., ac_k\}$, seja $\varphi(W, VM) = \{sched_1, sched_2, ..., sched_k\}$ o conjunto dos escalonamentos para execução paralela do conjunto de tarefas AC. O escalonamento de uma tarefa é definido como $sched(ac_i, VM_j, start, end)$, em que *start* é o tempo de início de execução de uma tarefa ac_i em uma dada VM_j e *end* o tempo fim.

Como definido em Oliveira *et al.* (2010), são feitos diversos escalonamentos *(sched)* durante toda a execução do *workflow*. Portanto é definido que $ord(sched_i)$ é a posição do $sched_i$ na ordem de sequência de todos os escalonamentos feitos, logo, temos $sched_i < sched_j \leftrightarrow ord(sched_i) < ord(sched_j)$.

Para executar o *workflow* é necessário criar um *cluster* virtual na nuvem, então considere $|VM|$ como o número de máquinas virtuais pertencentes a um *cluster* e $|AC|$ o número de atividades que devem ser executadas no *workflow* em questão. Agora podemos calcular o limite superior da taxa de processamento (*i.e. throughput*) das tarefas pertencentes a este *cluster* da seguinte maneira:

$$\beta = \frac{|VM| \times |AC|}{\sum_{i=0}^{k} P(ac_i, VM_j)} \tag{1}$$

A partir da equação 1, que representa a taxa de processamento de todas as tarefas, podemos agora definir o número de todas as tarefas que estão na fila para serem processadas como σ. Desta maneira, conseguimos definir o tempo de execução total do *workflow (makespan)* de todas as k tarefas de um dado *workflow* científico como sendo:

$$makespan = \frac{\sigma}{\beta} = \frac{\sum_{i=j}^{k} P(ac_i, VM_j)\sigma}{|VM| \times |AC|} \tag{2}$$

Com estas definições, conseguimos estimar os valores de tempo de execução, porém a primeira coisa que precisamos fazer antes de iniciar o escalonamento do *workflow* na nuvem é estimar o valor de $P(ac_i, VM_j)$, antes mesmo de iniciar o *makespan*. Para isso, podemos fazer uma média de $P(ac_i, VM_j)$ de todas as execuções anteriores e usar como entrada deste modelo de custo. No caso de não haver nenhuma execução anterior, deve-se usar uma estimativa extra modelo (Oliveira, 2012).

2.5.3 Transferência de dados

Transferência de dados é um custo muito importante, que deve ser contabilizado no custo total da execução do *workflow*, mas não é um objetivo que o usuário possa escolher

minimizar, ele apenas é calculado com base na transferência de arquivos de entrada e saída feitas entre as máquinas que executam uma tarefa de um *workflow* na nuvem. Como o SciCumulus lê e escreve em uma área compartilhada (Oliveira *et al.*, 2011), todos os escalonamentos possíveis em $\varphi(W, VM)$ leem e escrevem o mesmo volume de dados, logo, o custo sempre será o mesmo independente do escalonamento de um mesmo *workflow*.

Considere ac_i como uma tarefa associada a uma atividade $a_i(Act(ac_i) = a_i)$ de um *workflow* W com tempo de execução definido em $time(ac_w)$ e com $sched(ac_i, VM_k)$ já definido. Agora considere uma tarefa ac_j associada à atividade $a_j(Act(ac_j) = a_j)$ com dependência $DepAc(ac_i, ac_j, volume(ac_i, ac_j))$ e $sched(ac_j, VM_x)$, onde $volume(ac_i, ac_j)$ é a quantidade de dados transferidos entre ac_i e ac_j. Como $VM_k \neq VM_x$, é necessário incluir uma nova dependência de dados e uma tarefa artificial em W para representar a transferência de dados entre ac_i e ac_j. Portanto, vamos substituir $DepAc(ac_i, ac_j, volume($ $ac_i, ac_j))$ por $DepAc(ac_i, ac_w, volume(ac_i, ac_j))$ e faremos $DepAc(ac_w, ac_j, volume(ac_i, ac_j))$, em que ac_w é uma tarefa artificial criada para representar essa transferência. Além disso precisamos definir $sched(ac_w, VM_k)$ e $sched(ac_w, VM_x)$ para que ela possa ser executada. Logo teremos o tempo de execução da tarefa ac_w:

$$time(ac_w) = \frac{volume(ac_i, ac_j)}{min(network(VM_k), network(VM_x))} \tag{3}$$

Onde existe $sched(ac_i, VM_k)$ e $sched(ac_j, VM_x)$. Agora conseguimos calcular o tempo total de transferência de dados *(TT)* para todas as tarefas de W, assumindo que existam k atividades e portanto *k-1* transferências:

$$TT = \sum_{w=0}^{k-1} time(ac_w) \tag{4}$$

2.5.4 Confiabilidade

Este critério tem como objetivo a confiabilidade, muito importante devido ao ambiente de execução em nuvem utilizado neste trabalho. O ambiente em nuvem pode apresentar flutuações de desempenho ou até mesmo falhas *(i.e máquina reinicia, desliga, muda de servidor)* e todo o trabalho desenvolvido até o instante é perdido completamente. Ou seja, quando falamos que o custo de confiabilidade foi alto, significa que perdemos horas e horas de trabalho. O valor de falha de uma máquina virtual pode ser obtido como $F(VM_j), \forall\ VM_j \in VM$ com valor constante (Oliveira, 2012).

Portanto, $F(VM_j)$ obtém o valor de probabilidade de uma falha ocorrer em uma

determinada VM_j durante a execução do *workflow*. Esse valor pode ser obtido com base nos dados armazenados sobre uma VM_j, somente se for a mesma máquina das execuções anteriores, onde foram gravadas estas informações. Do contrário, se ela for uma nova máquina virtual, é atribuído o valor máximo possível de confiabilidade, que é 1, em uma abordagem otimista.

O custo de confiabilidade é o inverso da confiabilidade da execução do *workflow*. Portanto, uma vez que minimizamos o custo de confiabilidade, aumentamos a confiabilidade da execução como um todo. Da mesma forma, quando a execução tem pouca confiabilidade *(i.e. vários erros)* o custo sai alto. Desta maneira, o custo de confiabilidade de uma determinada atividade rodar em uma máquina é representado por $R(ac_i, VM_j)$ e pode ser definido como:

$$R(ac_i, VM_j) = F(VM_j) \times P(ac_i, VM_j) \tag{5}$$

Para que a confiabilidade se aproxime de 1, o valor de $R(ac_i, VM_j)$ deve ser minimizado. Analogamente, a confiabilidade geral do sistema é dada pela soma de todas as confiabilidades calculadas para todas as tarefas daquela execução. Considere então $R(VM_j)$ como sendo o conjunto de tarefas que já foram executadas em VM_j , logo a confiabilidade total de uma máquina pode ser calculada da seguinte forma:

$$R(VM_j) = \sum R(ac_i, VM_j), \ \forall \ ac_i \in rodados \ na \ VM_j \tag{6}$$

2.5.5 Custo Monetário

O objetivo de custo monetário é uma das principais motivações para se usar o ambiente de nuvem, pois o usuário não tem investimento inicial, apenas "aluga"máquinas por um período de tempo, pagando somente pelos recursos que consumir (Vaquero *et al.*, 2008), mas deve-se usar de forma consciente, do contrário pode acabar saindo caro da mesma forma. Conforme dito anteriormente, nesta dissertação foi adotado a nuvem da Amazon EC2, e logo ela possui seu esquema de especificação de custo que pode variar de provedor para provedor. No caso do provedor em questão, é cobrado a janela de tempo por hora, independente se a máquina está ociosa ou processando, o valor é o mesmo.

Vamos definir o custo monetário de execução como $M(\varphi)$. O tempo de execução total deve ser a soma das várias janelas de tempo de tamanho Qt o que nos leva ao conjunto $TQ = \{tq_1, tq_2, ..., tq_x\}$ de janelas de tempo, onde denominamos o valor da janela de tempo a ser pago como Vq. Considere $Exec(tq_z, VM_j)$ uma função que retorne se a VM_j está

executando uma tarefa na janela de tempo tq_z, desta forma:

$$Exec(tq_z, VM_j) \begin{cases} 1 \; se \; VM_j \; roda \; ac_i/ac_i \in Ac \; em \; tq_z \\ 0 \; caso \; contrário \end{cases} \tag{7}$$

Como a função acima só retorna positivo quando a máquina está processando, precisamos considerar também o tempo em que as máquinas ficam inativas aguardando a transferência de dados. Portanto, o cálculo do custo monetário fica divido em duas partes: a primeira calcula o custo enquanto as máquinas estão processando as tarefas. E a segunda calcula o tempo em que as máquinas aguardam o tempo de transferência dos dados. Logo teremos o seguinte:

$$M(\varphi) = (Vq \times \sum_{j=1}^{|VM|} \sum_{i=1}^{\frac{makespan}{|tq|}} Exec(tq_i, VM_j) + Vq \times \sum_{i=0}^{k} time(ac_j)) \tag{8}$$

Com isso obtemos o custo de execução, porém, ainda há outra variável a ser adicionada, que é a unidade de informação transferida pela rede, pois os provedores de nuvem também cobram por isso. Dessa forma, teremos o custo monetário de transferência como $TR(\varphi)$. Considere como Vt o valor que o usuário tem de pagar por cada unidade de dados transferidos (sem distinção). Portanto o valor total a ser pago pela transferência de dados é:

$$TR(\varphi) = (Vt \times \sum_{j=0}^{k-1} \sum_{i=0}^{k-1} volume(ac_i, ac_j)) \tag{9}$$

2.6 Algoritmo de Escalonamento

O algoritmo de escalonamento da abordagem de Oliveira (2012) implementada no SciCumulus, em que a solução representa um *trade-off* entre três critérios: custo monetário, tempo de execução e confiabilidade. Sua abordagem utiliza um algoritmo guloso, conforme mostra o Algoritmo 1. Onde é escolhida para cada ativação uma máquina ociosa mais adequada para execução.

Como entrada, o algoritmo precisa do conjunto de máquinas virtuais, o prazo para o *workflow* terminar e o orçamento máximo pretendido a ser gasto pelo usuário. Na linha 2 o escalonamento é inicializado, logo após, na linha 3 são carregadas as ativações disponíveis e que não tenham nenhuma dependência de dados associada a outra tarefa, portanto, estando disponíveis de imediato para serem executadas.

Algoritmo 1: **Escalonamento Guloso (Oliveira, 2012)**

Entrada:
W: o *workflow* científico
VM: conjunto de máquinas virtuais a serem usadas
Deadline: prazo para que o *workflow* termine
Budget: orçamento limite informado pelo cientista
Saída:
$\varphi(W, VM)$: escalonamento realizado de W em VM

1: *loadBalance(VM, Deadline, Budget)*
2: $\varphi(W, VM) \leftarrow \varnothing$
3: $available \leftarrow \{ca_i \in CA | \forall ca_j \in CA \, \neg DepT(ca_i, ca_j, ds)\}$
4: $ready \leftarrow \{vm_j \in VM | \forall vm_j \in VM \, | av(vm_j)\}$
5: **for each** vm in VM **do**
6: vm.partGranFactor $\leftarrow 1$
7: vm.execTime $\leftarrow \infty$
8: vm.prevExecTime $\leftarrow \infty$
9: vm.maxGranFactor $\leftarrow \infty$
10: **end for each**
11: **while** $available \neq \varnothing$ **do**
12: **for each** vm ready **do**
13: maxGranFactor $\leftarrow$ getMaxGranFactor(vm)
14: partGranFactor $\leftarrow$ getPartGranFactor(vm)
15: avgTime $\leftarrow$ getAvgTime(vm)
16: **if** $maxGranFactor \neq \infty$ **then**
17: $clusters \leftarrow group(vm, maxGranFactor, avgTime)$
18: **else**
19: $clusters \leftarrow group(vm, partGranFactor, avgTime)$
20: **end if**
21: **for each** x in cluster **do**
22: $cost \leftarrow f(x, vm)$
23: $possible \leftarrow possible + \{sched(x_i, VM_j, cost)\}$
24: **end for each**
25: $chosen \leftarrow min(possible)$
26: $\varphi(W, VM) \leftarrow \varphi(W, VM) + chosen$
27: $perform(chosen, Threshold)$
28: $ready \leftarrow ready - \{vm\} + \{vm_j \in VM | av(vm_j)\}$
29: $available \leftarrow \{ca_i \in CA | \forall ca_j \in CA \, \neg DepT(ca_i, ca_j, ds)\}$
30: **end for each**
31: **end while**
32: **return** $\varphi(W, VM)$;

Na linha 4 é carregada a lista de máquinas disponíveis no *cluster*. Nas linhas 5 a 10 são inicializadas as máquinas com as informações principais para agrupar as tarefas. Então é iniciado o *loop* (linha 11) onde é analisado se existem tarefas disponíveis, em caso positivo é iniciado uma busca por qual máquina virtual ociosa será encarregada de executar aquela tarefa (linha 12). A ordenação das tarefas é feita de acordo com a capacidade da máquina e do cenário escolhido pelo usuário, que atribui pesos à variável α_i tal que $(0 \leq \alpha_i \leq 1)$, portanto isso identifica qual cenário em questão (*i.e.* α_1 é atribuído o maior peso e representa o cenário que prioriza o tempo de execução, portanto o algoritmo irá ordenar as máquinas virtuais das mais potentes para as mais lentas).

Quando uma máquina virtual estiver ociosa é analisado se existe um fator de granularidade ou granularidade parcial atribuído àquela *vm* (linhas 13-15), indicando qual tamanho máximo do agrupamento que aquela máquina suporta sem perda de desempenho considerável. Estes fatores são utilizados para agrupar as tarefas disponíveis para execução e as encapsular em uma nova tarefa. Onde logo depois estes fatores são usados para identificar qual agrupamento de tarefas (variável *clusters*) essa tarefa deve ficar (linhas 16-20). De posse dessa informação, é calculado agora na linha 22 qual custo para se executar a tarefa em uma determinada máquina, depois na linha 23 essa possibilidade é somada ao conjunto de escalonamentos possíveis. Ao sair do laço de repetição, na linha 25, é escolhido o escalonamento com menor custo. Este escalonamento agora é somado ao conjunto total (linha 26). Nas linhas 28 e 29 a lista de máquinas e tarefas disponíveis são atualizadas e o processamento continua. Este algoritmo é executado durante o processamento do *workflow*, onde para cada tarefa ele é chamado novamente e retornado o escalonamento escolhido na linha 32.

Conforme podemos analisar, o Algoritmo 1 calcula a melhor máquina ociosa para executar determinada tarefa, com uma visão local, baseado nas restrições de execução escolhidos pelo usuário $(\alpha_1, \alpha_2, \alpha_3)$, e retorna para o *workflow* uma única solução, porém quanto mais objetivos tiver para serem otimizados, pior se torna o escalonamento.

A ideia agora é calcular um conjunto de soluções, utilizando uma abordagem hierárquica multiobjetivo, deixando o *workflow* mais flexível para tomar certas medidas. Para isso, foi estendido o Algoritmo 1 para construir um conjunto de escalonamentos à partir de uma lista ordenada de objetivos. Esse algoritmo e essa abordagem será melhor explicada no Capítulo 3.

Capítulo 3

Escalonamento hierárquico multiobjetivo de workflows em nuvens de computadores

Uma das tarefas mais desafiadoras e complexas no ciclo de vida de um *workflow* é como escalonar ativações para determinados recursos em um ambiente heterogêneo (Durillo *et al.*, 2012). De acordo com a problemática encontrada em tentar agregar e otimizar uma série de objetivos no momento de escalonar uma ativação em um *workflow* científico de larga escala, a proposta dessa dissertação é uma abordagem hierárquica que se utiliza de múltiplas funções objetivo e que atende a várias restrições escolhidas pelo usuário. Porém, não atribuímos pesos a cada objetivo a ser otimizado (Oliveira, 2012), ou tentamos otimizar determinada função e não violar as restrições das demais (Sakellariou *et al.*, 2007), ou encontrar uma única solução por meio de agregação de todos os objetivos em uma função analítica (Assayad *et al.*, 2004; Garg *et al.*, 2009; Hakem e Butelle, 2007). A abordagem aqui proposta baseia-se nas ideias de Durillo *et al.* (2015), que busca encontrar um conjunto de soluções para cada atividade a ser executada no fluxo de dependência de tarefas.

3.1 Função Objetivo

Conforme a problemática encontrada ao se escalonar uma tarefa visando múltiplos objetivos e utilizando várias funções objetivos, como forma de consolidar a solução proposta nessa dissertação foi adicionado um novo objetivo, consumo energético, ao formalismo apresentado na seção 2.5 do Capítulo 2.

3.1.1 Consumo Energético

O consumo energético é uma nova preocupação que vem sendo agregada aos *SGWfC* nos últimos anos, buscando atender não só as necessidades do cientista, mas também a questões ambientais e de administradores de sistemas (Durillo *et al.*, 2014; Fard *et al.*, 2012). Segundo Hamilton (2009) 19% do orçamento total da Amazon em um período de quinze anos foi gasto com consumo energético e 23% com sistemas de refrigeração. Devido a estes fatores, foi considerado nesta dissertação a métrica de consumo energético e foi acoplado ao escalonador do SciCumulus.

Considere $CPUCons(VM_j)$ como uma função que retorna o consumo da Unidade Central de Processamento *(CPU - do inglês Central Processing Unit)* por hora *(watts/hora)* de uma determinada VM_j, analogamente a função $RAMCons(VM_j)$ retorna o consumo da memória de acesso aleatório *(RAM - do inglês Random Access Memory)* de uma VM_j em *(watts/hora)*. As informações sobre a *CPU* e *RAM* de cada máquina são fornecida pelo próprio sítio da Amazon EC2, e com a referência de cada item é possível verificar o consumo energético no site do fabricante, que no caso é a Intel[1].

Agora considere que se uma máquina VM_j é escalonada em $sched_i$ conseguiremos obter essa informação por meio da função:

$$isSched(ac_i, VM_j) = \begin{cases} 1 \ se \ \ sched(ac_i, VM_j) \\ 0 \ caso \ \ contrário \end{cases}$$

Com isso, conseguimos agregar ao custo total de consumo energético por meio da soma de todos os tempos das tarefas pertencentes a todas as atividades escalonadas nas máquinas do *workflow* W, da seguinte forma:

$$\sum_{i=0} \sum_{j=0} P(ac_i, VM_j) \times isSched(ac_i, VM_j) \tag{1}$$

Portanto, o consumo energético calculado nessa dissertação é realizado em duas partes, onde a primeira parte calcula o consumo da *CPU* de uma VM_j durante o processamento de uma ac_i, e a segunda parte calcula o consumo da memória *RAM* durante o processamento da máquina na execução de uma tarefa. Desta forma, basta usar a equação 1 e conseguimos somar o custo total de consumo energético $E(\varphi)$ de cada VM_j durante o

[1]http://www.intel.com.br/content/www/br/pt/homepage.html

processamento de todas as tarefas, como segue em:

$$E(\varphi) = \sum_{i=0} \sum_{j=0} CPUCons(VM_j) \times P(ac_i, VM_j) \times isSched(ac_i, VM_j) + RAMCons(VM_j) \times P(ac_i, VM_j) \times isSched(ac_i, VM_j) \quad (2)$$

3.2 Abordagem proposta

O algoritmo de escalonamento aqui proposto estende a abordagem de Oliveira (2012) implementada no SciCumulus, como forma de instanciar a solução proposta nessa dissertação. A ideia é criar um escalonamento para cada objetivo da lista ordenada recebida pelo usuário, formando portanto uma lista de escalonamentos.

O Algoritmo 2 mostra como é realizado o novo escalonamento. Inicia-se carregando a lista de tarefas disponíveis (linha 2). Nas linhas 3-5 é inicializado cada escalonamento do conjunto de escalonamentos como vazio. Após isso, assim como no algoritmo do SciCumulus (mostrado no Capítulo 2), as informações que serão utilizadas para gerenciar as máquinas são inicializadas (linhas 6-11). O *loop* principal começa a processar na linha 12, com a condição de haver tarefa disponível para ser escalonada. É importante ressaltar agora, que na linha 13 é feito o laço de repetição onde será feito um escalonamento diferente para cada tipo de objetivo presente na lista de objetivos recebido como entrada do *workflow* e então é criado uma lista temporária *C'* e inicializada como vazio. Também é carregado a lista de máquinas disponíveis para serem utilizadas naquele objetivo.

O próximo trecho (linhas 13-16) carrega as informações fundamentais de cada máquina, em que é analisado se existe um fator de granularidade ou granularidade parcial atribuído àquela *vm*. Este fatores são utilizados para agrupar as máquinas que são colocadas na variável *cluster*, de acordo com o fator de granularidade (linhas 19-23) e posteriormente são usadas para calcular o custo (linha 27) da tarefa com o objetivo *g* naquela *vm*. A lista temporária *C'* é então atualizada com a adição do novo escalonamento (linha 28).

Após calcular o custo de se executar a tarefa em cada uma das máquinas e adicionar aquele escalonamento à lista temporária, é então adicionado ao grupo de escalonamento pertencente àquele objetivo (linha 30) e ordenado em relação ao custo do menor para o maior (linha 31). Após terminar de montar os vários escalonamentos para determinada tarefa, ela então é subtraída da lista de tarefas disponíveis (linha 33). O algoritmo finaliza retornando o conjunto de escalonamentos montados para cada tarefa a ser executada.

Vale lembrar que este algoritmo é o método responsável por montar a lista de esca-

Algoritmo 2: **Escalonamento Hierárquico Multiobjetivo**

Entrada:
W: o *workflow* científico
VM: conjunto de máquinas virtuais a serem usadas
G: lista ordenada de objetivos escolhidos pelo usuário
Saída:
C: conjunto de escalonamentos

1: $available \leftarrow \{ca_i \in CA | \forall ca_j \in CA \neg DepT(ca_i, ca_j, ds)\}$
2: **for each** c in C **do**
3: c $\leftarrow \varnothing$
4: **end for each**
5: **for each** vm in VM **do**
6: vm.partGranFactor $\leftarrow 1$
7: vm.execTime $\leftarrow \infty$
8: vm.prevExecTime $\leftarrow \infty$
9: vm.maxGranFactor $\leftarrow \infty$
10: **end for each**
11: **while** $available \neq \varnothing$ **do**
12: **for each** g in G **do**
13: $C' \leftarrow \varnothing$
14: $ready \leftarrow \{vm_j \in VM | \forall vm_j \in VM | av(vm_j)\}$
15: **for each** vm in ready **do**
16: maxGranFactor $\leftarrow$ getMaxGranFactor(vm)
17: partGranFactor $\leftarrow$ getPartGranFactor(vm)
18: avgTime $\leftarrow$ getAvgTime(vm)
19: **if** $maxGranFactor \neq \infty$ **then**
20: $cluster \leftarrow group(vm, maxGranFactor, avgTime)$
21: **else**
22: $cluster \leftarrow group(vm, partGranFactor, avgTime)$
23: **end if**
24: $ready \leftarrow ready - \{vm\} + \{vm_j \in VM | av(vm_j)\}$
25: **end for each**
26: **for each** x in cluster **do**
27: $cost \leftarrow f(x, g, vm)$
28: $C' \leftarrow C' + \{sched(x_i, vm_j, cost)\}$
29: **end for each**
30: $C_g \leftarrow C_g + \{C'\}$
31: $sortMin(C_g)$
32: **end for each**
33: $available \leftarrow \{ca_i \in CA | \forall ca_j \in CADepT(ca_i, ca_j, ds)\}$
34: **end while**
35: **return** C;

lonamentos. A partir desta lista o *workflow* poderá escolher a máquina de menor custo para executar uma tarefa.

Com a lista de escalonamentos montada, para escolher a máquina de menor custo, o *workflow* irá calcular se a diferença do custo de executar a tarefa na primeira máquina for inferior a um *threshold* de 10% do custo da segunda máquina, será então utilizado o próximo critério e passado para a segunda lista de escalonamento na hierarquia e feito novamente a comparação, e assim sucessivamente até encontrar a máquina de menor custo. É importante dizer que o *threshold* é configurável, podendo ser aumentado ou diminuído pelo usuário.

Suponhamos que o cientista escolha a seguinte lista de prioridades para minimização dos critérios: consumo de energia, custo financeiro, confiabilidade e tempo de execução, como apresentado na Figura 3.1. Seguindo essa ordem, será gerado inicialmente um escalonamento atendendo ao primeiro objetivo e calculado utilizando o modelo de custo associado. A partir do melhor valor da função objetivo, são selecionados todos os escalonamentos cujo custo difira de no máximo 10% do melhor custo. Esses escalonamentos são considerados como "empates" e a partir daí é utilizado o segundo critério para gerar uma nova lista de escalonamentos. Essa ideia é similar à ordenação imposta por SGBDs na cláusula ORDER BY, onde uma série de critérios de ordenação são definidos de maneira hierárquica.

Portanto, conforme a Figura 3.1 o custo da primeira *vm* do escalonamento feito para "energia" foi menor que 10% em relação a segunda *vm*, logo *SGWfC* tem autonomia para transitar entre os escalonamentos feitos entre as métricas escolhendo sempre o menor custo de cada um deles se tiver uma diferença significativa. O escalonador passa para o escalonamento feito para custo financeiro, restringindo seu conjunto de comparações as duas máquinas que ficaram "empatadas". Como a diferença entre as duas máquinas escolhidas calculando o custo financeiro foi maior que o *threshold* escolhido, a máquina com menor custo que vai executar a tarefa seria então a primeira do escalonamento financeiro.

Uma restrição imposta na heurística proposta, é mostrada na Figura 3.2. Suponhamos que as duas primeiras máquinas da lista de "energia" fiquem "empatadas" no *threshold*, portando essas mesmas duas máquinas serão comparadas no próximo nível da hierarquia que é o de custo financeiro e novamente fiquem no mesmo *threshold*, e assim segue até que alcance o último nível da hierarquia e ainda precisasse transitar para outro critério, pois os escalonamentos ainda se encontram "empatados". Seria então escolhido a primeira máquina do primeiro escalonamento (e desconsiderado os 10% de diferença),

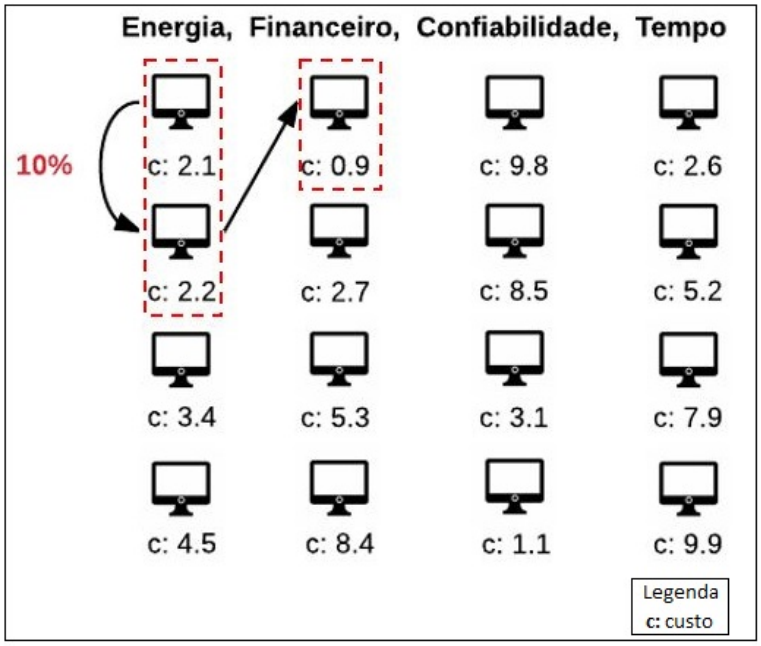

Figura 3.1: Exemplo de escalonamento hierárquico

i.e. o algoritmo tenta minimizar o máximo de critérios possível, porém caso não seja possível, o primeiro critério da lista é o escolhido. Assim, no pior caso, o algoritmo pode "degenerar" para uma otimização de um único objetivo.

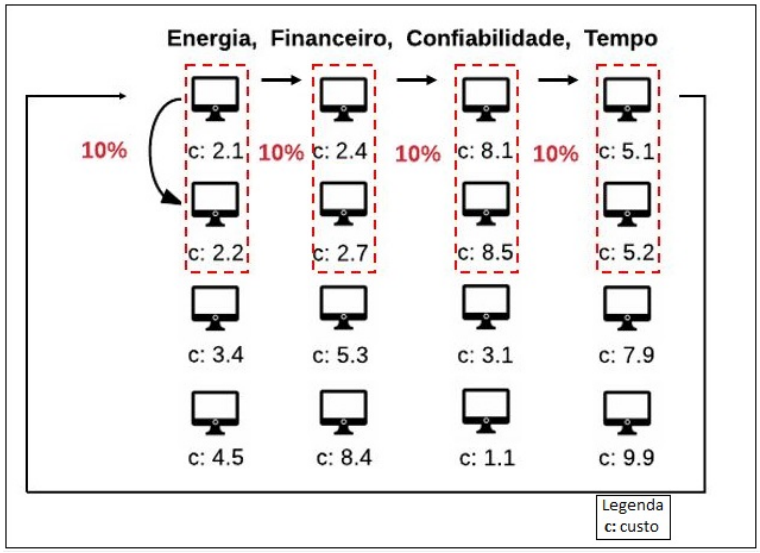

Figura 3.2: Exemplo de restrição de escalonamento hierárquico

No próximo Capítulo apresentamos como a abordagem proposta foi avaliada utilizando-se *workflows* sintéticos e reais da área de biologia.

Capítulo 4

Avaliação Experimental

Este capítulo faz uma avaliação da abordagem proposta, explicando as ferramentas adotadas para execução, e as métricas utilizadas, bem como os parâmetros e ambientes onde foram executados os testes. Para análise desta dissertação, foi utilizado o SGWfC SciCumulus (Oliveira *et al.*, 2010, 2012b) para gerenciar a execução do SciPhy (Ocaña *et al.*, 2011), que é um *workflow* de análise filogenética, e que será explicado na subseção 4.3. O ambiente para execução dos testes foi a nuvem da Amazon EC2, onde foi executado em paralelo, tratado na subseção 4.4 *Workflow* Real. Também foram feitos experimentos com um *Workflow* Sintético em um ambiente simulado baseado no CloudSim, tratado na subseção 4.2 *Workflow* Sintético. Os resultados obtidos com base nos comparativos feitos nas subseções 4.2 e 4.4 mostram uma melhora significativa com abordagem proposta.

4.1 Metodologia do experimento

Foram realizadas quatro execuções na abordagem proposta e na abordagem antiga do SciCumulus, tanto no *workflow* sintético como no *workflow* real, onde os objetivos estavam ordenados conforme o quadro de execuções:

1. Priorizando o Tempo de Execução: Modelo multi-objetivo (Peso 0.7 para objetivo tempo de execução e 0.1 para demais objetivos) x Lista de hierarquia: Tempo de execução, Confiabilidade, Custo financeiro, Consumo de energia.

2. Priorizando a Confiabilidade: Modelo multi-objetivo (Peso 0.7 para objetivo confiabilidade e 0.1 para demais objetivos) x Lista de hierarquia: Confiabilidade, Tempo de execução, Consumo de energia, Custo financeiro.

3. Priorizando o Custo financeiro: Modelo multi-objetivo (Peso 0.7 para objetivo custo financeiro e 0.1 para demais objetivos) x Lista de hierarquia: Custo monetário, Consumo de energia, Confiabilidade, Tempo de processamento.

4. Priorizando o Custo energético: Modelo multi-objetivo (Peso 0.7 para objetivo consumo de energia e 0.1 para demais objetivos) x Lista de hierarquia: Consumo de energia, Custo monetário, Confiabilidade, Tempo de processamento.

4.2 *Workflow* Sintético

Como forma de avaliar a abordagem proposta nesta dissertação, foram realizados dois experimentos. O primeiro deles foi utilizando um *workflow* sintético cujas informações de ativações foram inspiradas em perfis de aplicações reais de dinâmica de fluidos computacional. Além disso, foi necessário especificar os tipos de máquinas que processariam as ativações, e então foram atribuídas as mesmas configurações de um *cluster* com quatro máquinas da Amazon EC2[1], com sistema operacional Linux, segundo as informações em sua página. O *cluster* sintético foi montado conforme mostra a Tabela 4.1. Esse ambiente foi simulado em um computador com processador Intel i3-4010U 1.7 *GHz*, com 8 *GB* de memória *RAM*.

Quantidade	**Tipo**	**Descrição**
1	*M3.Medium*	Processador de 1 núcleo, com 3,75 GB de RAM
1	*M3.Large*	Processador com 2 núcleos e 7,5 GB de RAM
1	*M3.xLarge*	Processador com 4 núcleos e 15 GB de RAM
1	*M3.2xLarge*	Processador com 8 núcleos e 30 GB de RAM

Tabela 4.1: Lista de Máquinas *Cluster* Sintético

O *workflow* sintético é composto por 595 ativações (uma para cada arquivo de entrada), que possuem dependência entre si, divididos em 5 fases de processamento (simulando 5 programas de um *workflow* real de dinâmica de fluidos computacional). Uma ativação pode ser melhor paralelizada em uma máquina dependendo da sua quantidade de núcleos disponíveis. As quatro execuções e os pesos atribuídos aos objetivos foram definidos na seção 4.1.

Conforme apresentado a Figura 4.1, que contém o tempo das quatro execuções, a Abordagem antiga representa o escalonador com pesos e Abordagem Nova representa a abordagem proposta nessa dissertação. O primeiro conjunto de barras mostra que a nova

[1]https://aws.amazon.com/pt/ec2/instance-types/

abordagem terminou sua execução em menos tempo, sendo portanto superior à abordagem com pesos (antiga).

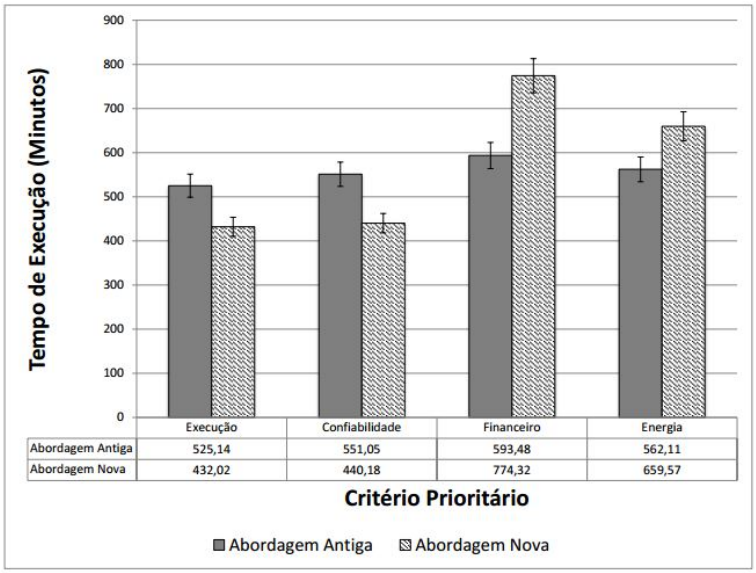

Figura 4.1: Gráfico Tempo de Execução *Workflow* Sintético

A segunda execução, conforme mostra o segundo conjunto de barras (confiabilidade) da Figura 4.1, novamente a abordagem proposta completou a execução do *workflow* mais rapidamente que a abordagem anterior. Isso acontece devido ao fato de as melhores máquinas (mais pontentes) serem também as mais confiáveis (menos propícias a erros). Vale a pena dizer que como o *workflow* sintético não é rodado na nuvem, e sim em uma máquina local, não ocorreu nenhum erro em nenhuma execução, portanto outro critério que poderia ser usado como comparação seria o tempo de execução.

A Figura 4.2 representa o consumo financeiro em dólares das quatro abordagens. Na terceira execução, que prioriza custo financeiro, a abordagem proposta foi superior a antiga, com custo monetário inferior, em que o cientista precisaria investir apenas 3,35 dólares e pela abordagem antiga seria necessário 4,47 dólares, conforme mostra o terceiro grupo do gráfico 4.2. Os valores calculados para consumo monetário foram baseados no modelo de custo definido no Capítulo 2 subseção 2.5.5.

A quarta e última execução do *workflow* sintético priorizou o consumo de energia gasto segundo o modelo de custo explicado na subseção 3.1.1 do Capítulo 3 e a nova abordagem teve melhor desempenho. A Figura 4.3 mostra o consumo de energia das quatro execuções, e pelo quarto conjunto de barras é possível notar que o escalonamento feito pela hierarquia da nova abordagem consumiu menos energia que o escalonador com pesos.

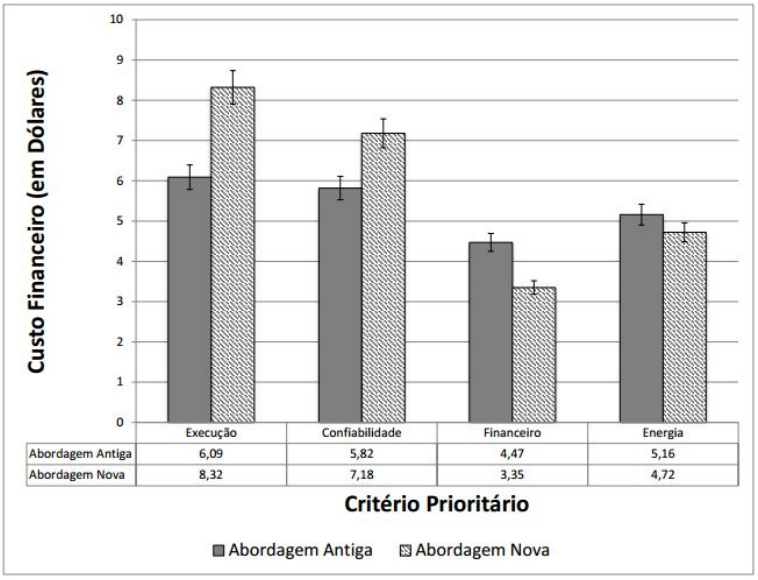

Figura 4.2: Gráfico Custo Financeiro *Workflow* Sintético

Considerando todas as abordagens executadas no *workflow* sintético, pode-se perceber então que um melhor escalonamento e consequente uso de recursos disponíveis é feito pelo escalonador proposto nessa dissertação, em que ele foi superior em todas as execuções.

4.3 SciPhy

O SciPhy é um *workflow* criado para gerar árvores filogenéticas, cujo objetivo é representar a semelhança entre os organismos, com base em sequências de aminoácidos, RNA ou DNA embutidos em arquivos de entrada no formato multi-fasta. Desta forma, o SciPhy auxilia a explorar a filogenia e determinar a vida evolutiva de genes ou genomas destes organismos (Ocaña *et al.*, 2011). Este *workflow* é composto por cinco atividades principais em sua versão 2.0 (Ocana *et al.*, 2015):

1. *DataSelection* (Seleção de dados e formatação)
2. *Mafft* (Construção do alinhamento genético)
3. *ReadSeq* (Conversão do formato de alinhamento)
4. *ModelGenerator* (Eleição do modelo evolutivo)
5. *RaXML* (Geração da árvore)

A primeira atividade (*dataselection*), que é implementada por um *script*, verifica a qualidade e o formato dos arquivos de entrada multi-fasta, sendo a operação mais rápida

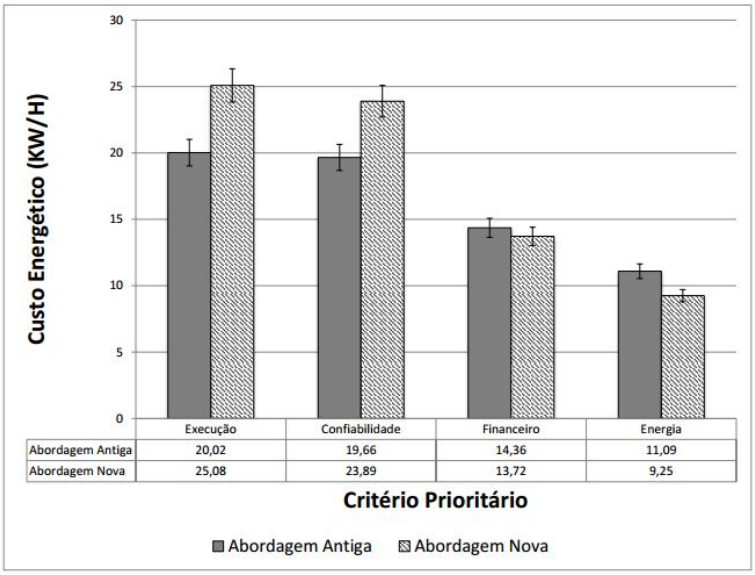

Figura 4.3: Gráfico Custo Energético *Workflow* Sintético

do *workflow*. A segunda atividade (*mafft*) faz a construção do alinhamento genético (MSA), recebendo como entrada os arquivos multi-fasta, utilizando o programa MAFFT. Existem mais outros quatro programas que podem ser utilizados nesta fase (*e.g.* Kalign, ClustalW, Muscle, ProbCons) para produção do MSA. A próxima atividade é o *readseq*, que converte os arquivos de formato FASTA para o formato PHYLIP, que é o esperado nas próximas atividades. Na atividade de *modelgenerator*, que é uma das mais demoradas no sistema, é feito o teste em cada um dos arquivos de entrada produzidos pela atividade anterior, com o objetivo de encontrar o melhor modelo evolutivo a ser utilizado. Por fim, tanto o melhor modelo evolutivo quanto o MSA associado a ele são utilizados na tarefa de geração de árvore filogenética, *raxml*. Como o cientista ainda não sabe qual método de alinhamento irá produzir a melhor árvore, ele tem de executar o *workflow* várias vezes, fazendo uma varredura de parâmetros e explorando diversos algoritmos de alinhamento possíveis. Mais informações a respeito do SciPhy podem ser obtidas em Ocana *et al.* (2015).

4.4 *Workflow* Real

Similarmente ao *workflow* sintético apresentado na seção 4.2, foram realizados testes com o *workflow* real SciPhy, descrito na seção 4.3, com 250 arquivos multi-fasta de entrada. O SciPhy foi executado no ambiente de nuvem da Amazon EC2, porém nesse experimento utilizamos um *cluster* com 9 máquinas de diferentes tipos totalizando 32 núcleos computacionais, conforme apresentado na Tabela 4.2.

Quantidade	Tipo	Descrição
2	*M3.Medium*	Processador de 1 núcleo, com 3,75 GB de RAM
3	*M3.Large*	Processador com 2 núcleos e 7,5 GB de RAM
2	*M3.xLarge*	Processador com 4 núcleos e 15 GB de RAM
2	*M3.2xLarge*	Processador com 8 núcleos e 30 GB de RAM

Tabela 4.2: Lista de Máquinas *Cluster* Real

De forma a avaliar a abordagem proposta, o SciPhy foi executado no SciCumulus variando-se os quatro objetivos e para cada um deles o *workflow* foi executado com as duas versões do escalonador. As quatro execuções em cada abordagem foram configuradas conforme descrito na seção 4.1, em relação a prioridade de objetivos.

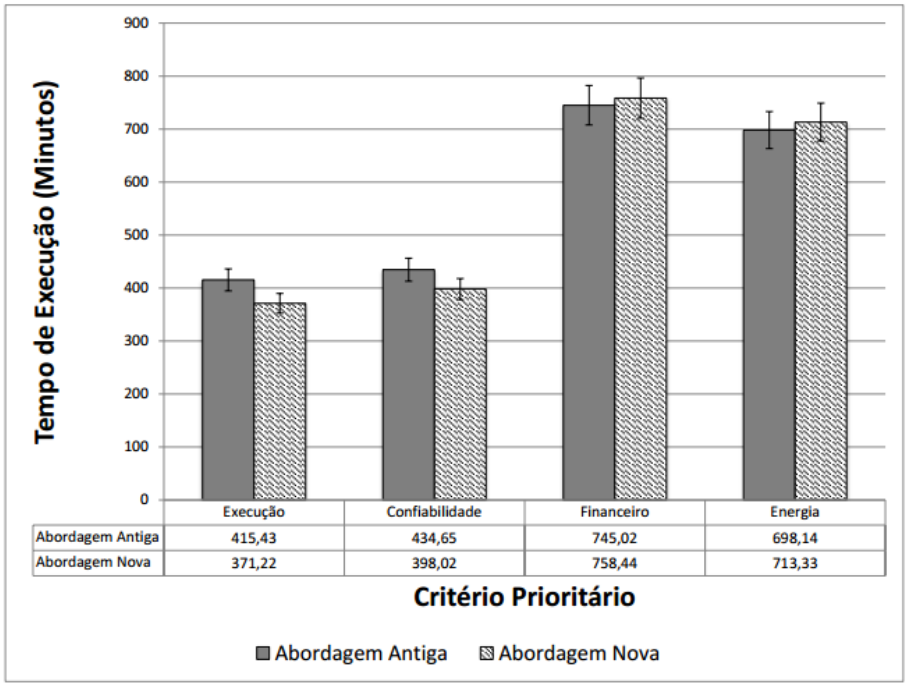

Figura 4.4: Gráfico Tempo de Execução *Workflow* Real

A primeira análise realizada foi em relação ao tempo de execução de cada uma das abordagens em cada um dos cenários propostos (*i.e.* prioridade dos critérios). A Figura 4.4 apresenta os tempos de execuções dos quatro cenários, em que o eixo vertical é representado por tempo em minutos e o eixo horizontal os critérios de prioridade. Conforme o primeiro conjunto de barras (tempo de execução como critério prioritário), nota-se que a abordagem proposta levou apenas 371 minutos para concluir a execução do *workflow* enquanto ao utilizar o escalonador com pesos da abordagem antiga levou 415 minutos. Portanto, a abordagem proposta teve melhor desempenho. Podemos perceber também que mesmo quando o critério principal é a confiabilidade, também foi possível reduzir o tempo de execução, uma vez que as máquinas mais confiáveis são as mais rápidas. Já para

os critérios de custo financeiro e custo de energia, o tempo de execução aumentou, uma vez que as máquinas com menor custo e menor consumo de energia são as mais lentas (*i.e* M3.medium).

Além do tempo de execução foram avaliadas a quantidade de falhas ocorridas em cada um dos cenários. A quantidade de falhas por execução do *workflow* real pode ser conferida com base no Gráfico apresentado na Figura 4.5. O cenário em que a confiabilidade era o critério primordial apresentou a mesma taxa de falhas na abordagem nova e na abordagem antiga. Porém, como o tempo de execução foi reduzido (Figura 4.4), podemos perceber que a nova abordagem é mais interessante, pois manteve a taxa de falhas e reduziu o tempo de execução. Em todos os outros cenários, a taxa de falhas aumentou com a utilização da abordagem nova, porém no caso em que o tempo de execução era o critério prioritário uma das falhas foi proporcionada pela própria Amazon, o que fez com que o escalonamento na abordagem nova fosse pior em relação a confiabilidade.

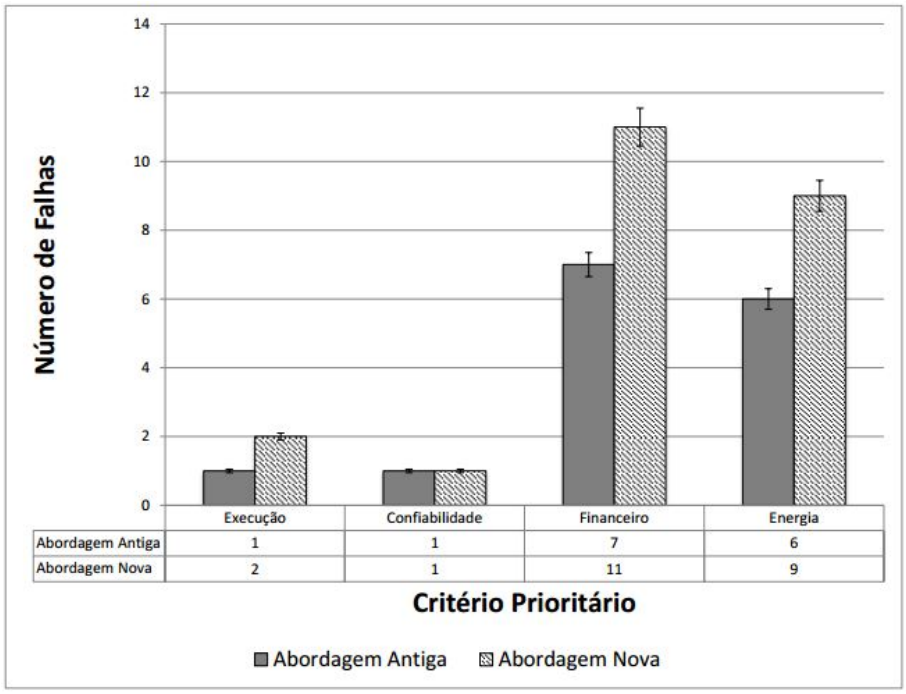

Figura 4.5: Gráfico Confiabilidade *Workflow* Real

A terceira análise é relativa ao custo financeiro de cada execução. Basicamente quando o cientista opta por minimizar prioritariamente o custo financeiro, ele está abrindo mão de desempenho e confiabilidade. Assim, a execução do *workflow* se caracteriza como uma execução mais longa e com mais falhas, pois máquinas mais baratas (e menos computacionalmente poderosas) serão usadas prioritariamente. Logo, se analisarmos somente o cenário em que o custo financeiro é o prioritário pela Figura 4.4 nota-se no terceiro grupo

que sua execução foi mais demorada, e para levar mais tempo, as máquinas mais inferiores foram as mais utilizadas e por isso a ocorrência de falhas é mais presente (conforme já apresentado na Figura 4.5). Entretanto como podemos observar na Figura 4.6, o custo financeiro desse foi o menor (nas abordagens antiga e nova) e foi efetivamente reduzido utilizando-se a nova abordagem. Pode-se peceber que mesmo quando o critério de consumo energético era o prioritário, a nova abordagem ainda foi capaz de reduzir o custo financeiro associado.

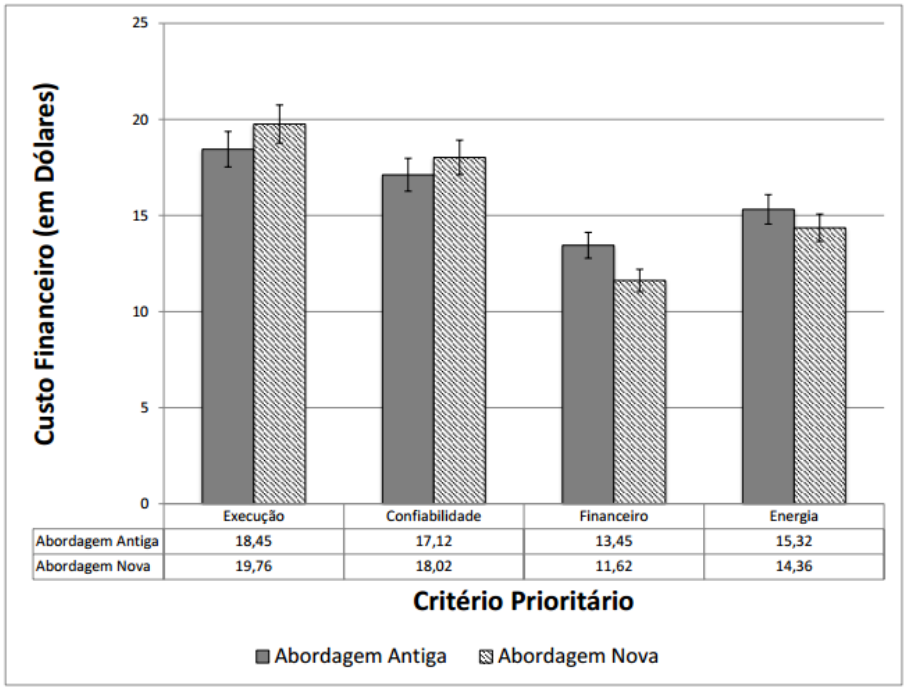

Figura 4.6: Gráfico Custo Financeiro *Workflow* Real

A quarta e última análise tem como objetivo verificar o consumo de energia do *cluster* virtual durante a execução do *workflow*. Os valores de consumo energético podem ser obtidos através da Figura 4.7. As máquinas que consomem menos energia (e para onde as maiores ativações são alocadas quando se quer minimizar o custo de energia) são as com menor poder de processamento, consequentemente essas execuções tem seu tempo total como o segundo mais demorado, e a segunda maior ocorrência de falhas. Porém, ao analisarmos o consumo de energia somente, como apresentado na Figura 4.7 podemos peceber que quando o critério primordial era o consumo energético, a abordagem proposta consumiu menos *kw/h* que a abordagem com pesos. Portanto, alcançando melhor o objetivo proposto.

Conforme os resultados obtidos, a nova abordagem teve melhores resultados em todas as execuções no *workflow* real, comprovando que este tipo de escalonamento consegue ter

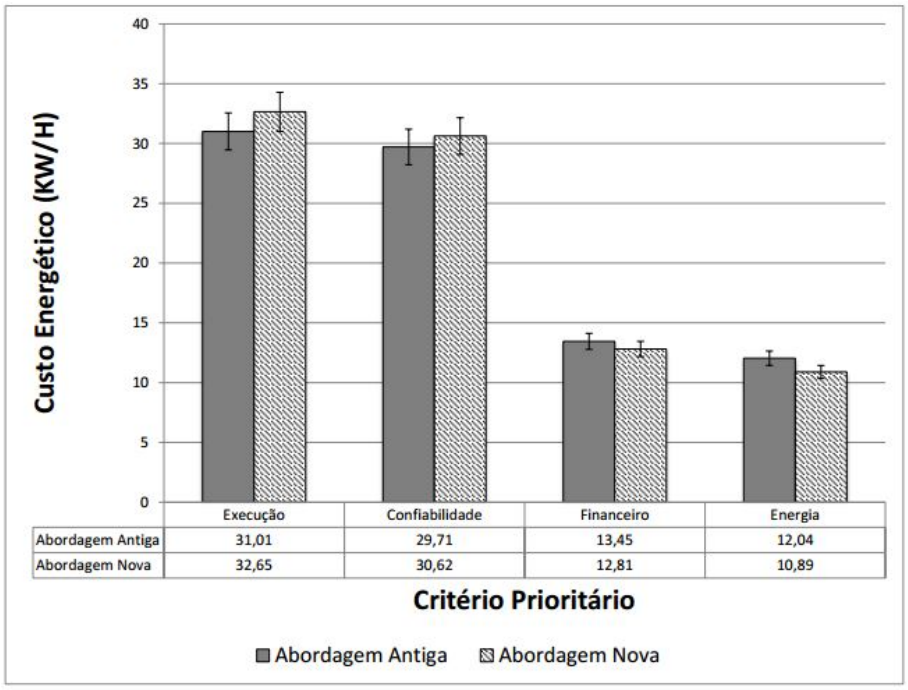

Figura 4.7: Gráfico Consumo Energético *Workflow* Real

melhores resultados do que uma abordagem que utiliza pesos em escalonamento multi-objetivos, atendendo melhor as necessidades do usuário e proporcionando um ambiente mais real, com um *workflow* mais eficiente.

Capítulo 5

Trabalhos Relacionados

Este capítulo aborda um estudo feito sobre os trabalhos relacionados na comunidade científica, a proposta dessa dissertação, que consta em abordagens de escalonamento multiobjetivo em nuvem de computadores, porém também leva-se em consideração os trabalhos que relacionam ambientes heterogêneos para execução de *workflows* científicos em larga escala.

Grande parte dos trabalhos na comunidade científica acadêmica busca soluções para otimizar apenas um objetivo (Zhao *et al.*, 2007; Deelman *et al.*, 2007; Altintas *et al.*, 2004; Hull *et al.*, 2006; Callahan *et al.*, 2006), alguns outros otimizam para dois objetivos (Pandey *et al.*, 2010; Garg *et al.*, 2009; Assayad *et al.*, 2004; Hakem e Butelle, 2007; Canon *et al.*, 2011; Sakellariou *et al.*, 2007; Yu *et al.*, 2007; Durillo *et al.*, 2012, 2013, 2014, 2015; Mezmaz *et al.*, 2011), e uma quantidade bem pequena atende três ou mais objetivos (Fard *et al.*, 2012, 2014). Apenas um trabalho mostrou uma abordagem diferenciada (Durillo *et al.*, 2015), utilizando os mesmos conceitos dessa dissertação, porém atendendo apenas tempo de execução e custo financeiro, e também diferencia-se em relação ao ambiente de trabalho, em que o seu foco é a federação de nuvem.

Garg *et al.* (2009) utiliza um escalonamento buscando otimizar *makespan* e confiabilidade, através de um vetor de pesos. Assayad *et al.* (2004) e Hakem e Butelle (2007) utilizam abordagens similares para otimizar confiabilidade e tempo. A principal desvantagem dessas abordagens é que a solução depende de uma combinação de pesos entre os objetivos para tentar atender as necessidades do usuário, que na maioria das vezes nem conhece como o *workflow* vai se comportar com aqueles dados de entrada, portanto terá que executar várias e várias vezes até ir conhecendo melhor e acertando aos poucos a distribuição de pesos dos objetivos.

Sakellariou *et al.* (2007) com dois objetivos e Fard *et al.* (2012) com quatro objetivos (*makespan*, consumo de energia, custo monetário, confiabilidade) calculam a solução otimizando um objetivo e tentando não violar as restrições dos outros baseado nos requisitos do usuário, porém essas aplicações foram feitas baseadas em *grids*. Fard *et al.* (2012) apesar de otimizar quatro objetivos só consegue fornecer apoio à nuvem para dois objetivos, otimizando os quatro objetivos faz um escalonamento estático, portanto não apropriado para nuvem.

Utilizando técnicas baseadas em algoritmos genéticos, Yu *et al.* (2007) busca otimizar *makespan* e custo financeiro. Na mesma linha de algoritmos genéticos ainda temos Mezmaz *et al.* (2011) utilizando meta-heurística para *makespan* e consumo energético em nuvens. Talukder *et al.* (2009) otimiza cofiabilidade, custo financeiro e *makespan*. Canon *et al.* (2011) utilizando heurística para algoritmo guloso atender confiabilidade e *makespan*. Durillo *et al.* (2013, 2015) utiliza um algoritmo denominado MOHEFT, para construir uma lista de escalonamentos, abordagem parecida com a dessa dissertação, porém apenas para *makespan* e custo energético, em (Durillo *et al.*, 2015) seu ambiente de execução é a federação de nuvens.

Pandey *et al.* (2010) utiliza uma heurística baseado em otimização de enxame de partículas (do inglês *particle swarm optimization - PSO*) para escalonamento em nuvens levando em consideração custo financeiro para processamento e transmissão, sua principal desvantagem é que sua métrica demora muito a convergir, gerando um alto custo computacional, pois não visa outros objetivos como por exemplo o tempo de execução.

Duan *et al.* (2014) faz escalonamento otimizando tempo de execução e custo financeiro, porém com opções de restrição de largura de banda e armazenamento, utilizando um algoritmo baseado em teoria de jogos. Esta abordagem foi desenvolvida para nuvens híbridas, onde existem no mínimo um nuvem privada e uma pública. A desvantagem é que exige do usuário o uso de uma nuvem privada e a quantidade de objetivos não atende a um cenário mais real.

O Swift/T (Zhao *et al.*, 2007), que é um tradicional SGWfC e que utiliza a máquina de *workflow* Karajan (Von Laszewski *et al.*, 2007), em seu escalonador os princípios básicos são tempo de execução total e confiabilidade, porém o usuário não define prioridade nem pesos, pois só existe essa otimização.

Outro SGWfC com grande visibilidade é o Pegasus (Deelman *et al.*, 2007), que foi inicialmente proposto para ambientes de *cluster* e *grids* mas sua versão mais recente já contempla execução em nuvem, sua otimização é focado apenas em tempo de execução,

sendo portanto um escalonador de objetivo único.

O escalonador do Kepler (Altintas *et al.*, 2004), considera apenas tempo de execução total, até mesmo em sua versão com a máquina de execução acoplada ao *Hadoop* (Wang *et al.*, 2009), não possuem dinamismo para considerar as características dispostas pela nuvem. Seguindo a mesma linhagem, o VisTrails (Callahan *et al.*, 2006), também acoplou sua máquina ao Hadoop (Howe *et al.*, 2009), porém também só considera *makespan*, assim como o Taverna (Hull *et al.*, 2006). A tabela 5.1 faz um resumo dos trabalhos relacionados abordados nessa dissertação.

Abordagem	**Parâmetros de Escalonamento**	**Apoio a Nuvem**
Swift (Zhao *et al.*, 2007)	*makespan*	Sim
Pegasus (Deelman *et al.*, 2007)	*makespan*	Sim
Kepler (Altintas *et al.*, 2004)	*makespan*	Não
Taverna (Hull *et al.*, 2006)	*makespan*	Não
Vistrails (Callahan *et al.*, 2006)	*makespan*	Não
Pandey *et al.* (2010)	custo financeiro	Sim
Garg *et al.* (2009)	*makespan* e confiabilidade	Não
Assayad *et al.* (2004)	*makespan* e confiabilidade	Não
Hakem e Butelle (2007)	*makespan* e confiabilidade	Não
Canon *et al.* (2011)	*makespan* e confiabilidade	Não
Sakellariou *et al.* (2007)	*makespan* e custo financeiro	Não
Yu *et al.* (2007)	*makespan* e custo financeiro	Não
Durillo *et al.* (2012, 2015)	*makespan* e custo financeiro	Sim
Mezmaz *et al.* (2011)	*makespan* e consumo energético	Sim
Durillo *et al.* (2013, 2014)	*makespan* e consumo energético	Não
Fard *et al.* (2012, 2014)	*makespan*, consumo energético, custo financeiro, confiabilidade	Parcial

Tabela 5.1: Comparativo de abordagens

Capítulo 6

Conclusão e Trabalhos Futuros

Neste capítulo são feitas as conclusões dessa dissertação, tratando as contribuições e limitações deparadas no desenvolver do trabalho, e também apresenta os trabalhos futuros que podem ser desenvolvidos dando continuidade ao projeto principal.

O escalonamento multiobjetivo de *workflows* em nuvens de computadores utilizando-se uma abordagem de pesos acaba passando por dois problemas principais. O primeiro é o desconhecimento do cientista ao rodar o experimento pela primeira vez sem saber como de fato o *workflow* vai se comportar com aqueles pesos atribuídos, como eles vão representar de fato suas necessidades. E o segundo problema é que ao agregar mais métricas a função objetivo, esta acaba perdendo o foco principal e os pesos se dissipam entre os vários objetivos.

Em frente a essa problemática, baseado nas ideias de Durillo *et al.* (2015, 2014, 2013, 2012) que utiliza uma abordagem denominada Pareto, em que é construído uma lista de escalonamentos como solução para atender a todos os objetivos do *workflow*, foi desenvolvido um escalonador hierárquico multiobjetivo nessa dissertação, em que é construído uma lista de escalonamentos baseado em todos os objetivos informados pelo cientista, em que ele não precisa atribuir pesos apenas escolhe a ordem de prioridade dos objetivos, e calculado o custo das tarefas em todas as máquinas para todos os objetivos, podendo escolher a melhor máquina do *cluster* para resolução daquela tarefa e quais as máquinas suplentes para executa-lá seguindo as restrições de prioridade passadas pelo usuário. Em que uma máquina escolhida pelo escalonador só deixa de executar a tarefa se ela não estiver ociosa e se o seu custo naquela métrica da hierarquia for maior que 10% da máquina suplente *(threshold)*. Dessa forma, consegue-se alcançar outras máquinas que tem custo inferior mas em relação a outra métrica e que fornece mais vantagem para a execução do *workflow*.

Como forma de validar essa solução proposta, foi utilizado o SciCumulus para instanciar essa hipótese. Portanto a abordagem discutida nessa dissertação baseou-se nos trabalhos e experiências anteriores do grupo de pesquisa na construção do SciCumulus (Oliveira, 2012; Oliveira *et al.*, 2010, 2012a, 2011), uma máquina de gerência de *workflow* científico de larga escala próprio para execução em ambientes de PAD, sendo adaptável aos recursos elásticos da nuvem em tempo de execução.

A abordagem anterior do escalonador do SciCumulus utilizava um conceito bem comum em se tratando de *workflows* científicos multiobjetivos, em que o cientista atribui pesos para cada objetivo, definindo assim o tipo de execução do *cluster* para que possa atender suas necessidades. Porém, como o SciCumulus otimizava somente 3 objetivos: tempo de execução, custo monetário e confiabilidade. Então tornou-se necessário a formulação de uma nova métrica a ser agregada ao escalonador: custo energético, que foi aqui desenvolvida para poder efetivar a proposta dessa dissertação.

Os resultados mostraram benefícios em se utilizar esse tipo de abordagem, em que pelas comparações feitas na avaliação experimental, a abordagem proposta foi superior em todas as métricas, provando portanto a hipótese dessa dissertação. Outra grande vantagem é que com a abordagem proposta, pode-se agora adicionar quantas métricas forem necessárias ou convenientes futuramente ao SciCumulus.

Algumas dificuldades encontradas no decorrer da dissertação foram na fase da avaliação experimental durante a execução do *workflow* real na nuvem, na configuração do ambiente e na execução que na maioria das vezes levava muito tempo para terminar e quando não obtínhamos os resultados desejados, devido a falhas na ativação das tarefas ou nas máquinas, por isso foi necessário reiniciar o processo algumas vezes.

Como trabalhos futuros seria interessante evoluir o SGWfC SciCumulus para ambiente de federação de nuvens, para que o usuário e a gerência do *workflow* possa ter mais opções de custo benefício nas máquinas fornecidas pelos provedores de nuvens, como foi feito no trabalho de Durillo *et al.* (2015). Adicionar mais métricas ao escalonador, como por exemplo taxa de transferência, tornando assim a solução mais robusta, também seria uma forma interessante de evolução.

Referências Bibliográficas

Ilkay Altintas, Chad Berkley, Efrat Jaeger, Matthew Jones, Bertram Ludascher, e Steve Mock. Kepler: an extensible system for design and execution of scientific workflows. In *International Conference on Scientific and Statistical Database Management*, pages 423–424, Santorini Island, Greece, 2004. IEEE.

Ismail Assayad, Alain Girault, e Hamoudi Kalla. A bi-criteria scheduling heuristic for distributed embedded systems under reliability and real-time constraints. In *International Conference on Dependable Systems and Networks*, pages 347–356, Florence, Italy, 2004. IEEE.

G Bruce Berriman, Ewa Deelman, John Good, Joseph C Jacob, Daniel S Katz, Anastasia C Laity, Thomas A Prince, Gurmeet Singh, e Mei-Hui Su. Generating complex astronomy workflows. In *Workflows for e-Science*, pages 19–38. Springer, 2007.

Cristina Boeres, Idalmis Milián Sardiña, e Lúcia MA Drummond. An efficient weighted bi-objective scheduling algorithm for heterogeneous systems. *Parallel Computing*, 37 (8):349–364, 2011.

Peter Buneman e Wang-Chiew. Tan. Provenance in databases. In *International Conference on Management of Data*, pages 1171–1173, New York, USA, 2007. ACM.

Steven P Callahan, Juliana Freire, Emanuele Santos, Carlos E Scheidegger, Cláudio T Silva, e Huy T Vo. Vistrails: visualization meets data management. In *International Conference on Management of Data*, pages 745–747, Chicago, USA, 2006. ACM.

Louis-Claude Canon *et al.* Mo-greedy: an extended beam-search approach for solving a multi-criteria scheduling problem on heterogeneous machines. In *International Symposium on Parallel and Distributed Processing Workshops and Phd Forum (IPDPSW)*, pages 57–69, Anchorage, USA, 2011. IEEE.

Yongyun Cho, Jongbae Moon, e Hyun Yoe. A context-aware service model based on workflows for u-agriculture. In *Computational Science and Its Applications–ICCSA 2010*, pages 258–268. Springer, 2010.

Ben Clifford, Ian Foster, Jens-S Voeckler, Michael Wilde, e Yong Zhao. Tracking provenance in a virtual data grid. *Concurrency and Computation: Practice and Experience*, 20(5):565–575, 2008.

Sérgio Manuel Serra da Cruz, Maria Luiza M Campos, e Marta Mattoso. Towards a taxonomy of provenance in scientific workflow management systems. In *World Conference on Services-I*, pages 259–266, Los Angeles, USA, 2009. IEEE.

Susan B Davidson e Juliana Freire. Provenance and scientific workflows: challenges and opportunities. In *International Conference on Management of Data*, pages 1345–1350, Vancouver, Canada, 2008. ACM.

Ewa Deelman, Gaurang Mehta, Gurmeet Singh, Mei-Hui Su, e Karan Vahi. Pegasus: mapping large-scale workflows to distributed resources. In *Workflows for e-Science*, pages 376–394. Springer, 2007.

Ewa Deelman, Dennis Gannon, Matthew Shields, e Ian Taylor. Workflows and e-science: An overview of workflow system features and capabilities. *Future Generation Computer Systems*, 25(5):528–540, 2009.

Jack J Dongarra, Emmanuel Jeannot, Erik Saule, e Zhiao Shi. Bi-objective scheduling algorithms for optimizing makespan and reliability on heterogeneous systems. In *Symposium on Parallel Algorithms and Architectures*, pages 280–288, San Diego, USA, 2007. ACM.

Rubing Duan, Radu Prodan, e Xiaorong Li. Multi-objective game theoretic schedulingof bag-of-tasks workflows on hybrid clouds. *IEEE Transactions on Cloud Computing*, 2 (1):29–42, 2014.

Juan J Durillo, Hamid Mohammadi Fard, e Radu Prodan. Moheft: A multi-objective list-based method for workflow scheduling. In *International Conference on Cloud Computing Technology and Science (CloudCom)*, pages 185–192, Taipei, Taiwan, 2012. IEEE.

Juan J Durillo, Vlad Nae, e Radu Prodan. Multi-objective energy-efficient workflow scheduling using list-based heuristics. *Future Generation Computer Systems*, 36:221–236, 2014.

Juan J Durillo, Radu Prodan, e Jorge G Barbosa. Pareto tradeoff scheduling of workflows on federated commercial clouds. *Simulation Modelling Practice and Theory*, 58:95–111, 2015.

Juan José Durillo, Vlad Nae, e Radu Prodan. Multi-objective workflow scheduling: An analysis of the energy efficiency and makespan tradeoff. In *International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, pages 203–210, Delft, Netherlands, 2013. IEEE.

Hamid Mohammadi Fard, Radu Prodan, Juan Jose Durillo Barrionuevo, e Thomas Fahringer. A multi-objective approach for workflow scheduling in heterogeneous environments. In *International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, pages 300–309, Ottawa, Canada, 2012. IEEE Computer Society.

Hamid Mohammadi Fard, Radu Prodan, e Thomas Fahringer. Multi-objective list scheduling of workflow applications in distributed computing infrastructures. *Journal of Parallel and Distributed Computing*, 74(3):2152–2165, 2014.

Renato Fileto, Ling Liu, Calton Pu, Eduardo Delgado Assad, e Claudia Bauzer Medeiros. Poesia: An ontological workflow approach for composing web services in agriculture. *The VLDB Journal-The International Journal on Very Large Data Bases*, 12(4):352–367, 2003.

Juliana Freire, David Koop, Emanuele Santos, e Cláudio T Silva. Provenance for computational tasks: A survey. *Computing in Science & Engineering*, 10(3):11–21, 2008.

W Freudling, M Romaniello, DM Bramich, P Ballester, V Forchi, CE García-Dabló, S Moehler, e MJ Neeser. Automated data reduction workflows for astronomy-the eso reflex environment. *Astronomy & Astrophysics*, 559:A96, 2013.

Saurabh Kumar Garg, Rajkumar Buyya, e Howard Jay Siegel. Scheduling parallel applications on utility grids: time and cost trade-off management. In *Australasian Conference on Computer Science-Volume 91*, pages 151–160, Wellington, New Zealand, 2009. Australian Computer Society, Inc.

Carole Goble, Chris Wroe, e Robert Stevens. The mygrid project: services, architecture and demonstrator. In *UK e-Science All Hands Meeting*, pages 595–602, Nottingham, UK, 2003.

Mourad Hakem e Franck Butelle. Reliability and scheduling on systems subject to failures. In *International Conference on Parallel Processing*, pages 38–38, Xian, China, 2007. IEEE.

James Hamilton. Cooperative expendable micro-slice servers (cems): low cost, low power servers for internet-scale services, 2009.

Bill Howe, Huy Vo, Claudio Silva, e J Freire. Query-driven visualization in the cloud with mapreduce. In *Workshop on Ultrascale Visualization*, Portland, Oregon, 2009.

Duncan Hull, Katy Wolstencroft, Robert Stevens, Carole Goble, Mathew R Pocock, Peter Li, e Tom Oinn. Taverna: a tool for building and running workflows of services. *Nucleic acids research*, 34(suppl 2):W729–W732, 2006.

Melissa Lemos, Marco A Casanova, Luiz Fernando Bessa Seibel, José Antonio Fernandes de Macedo, e Antonio Basílio de Miranda. Ontology-driven workflow management for biosequence processing systems. *Lecture notes in computer science*, pages 781–790, 2004.

W Martinho, E Ogasawara, D Oliveira, F Chirigati, I Santos, GHT Travassos, e M Mattoso. A conception process for abstract workflows: an example on deep water oil exploitation domain. In *International Conference on e-Science*, Oxford, UK, 2009. IEEE.

Marta Mattoso, Cláudia Werner, G Travassos, Vanessa Braganholo, e Leonardo Murta. Gerenciando experimentos científicos em larga escala. *SBC*, page 121, 2008.

Marta Mattoso, Cláudia Werner, G Travassos, Vanessa Braganholo, Leonardo Murta, Eduardo Ogasawara, F Oliveira, e Wallace Martinho. Desafios no apoio à composição de experimentos científicos em larga escala. *Seminário Integrado de Software e Hardware, SEMISH*, 9:36, 2009.

Marta Mattoso, Claudia Werner, Guilherme Horta Travassos, Vanessa Braganholo, Eduardo Ogasawara, Daniel Oliveira, Sergio Cruz, Wallace Martinho, e Leonardo Murta. Towards supporting the life cycle of large scale scientific experiments. *International Journal of Business Process Integration and Management*, 5(1):79–92, 2010.

Mohand Mezmaz, Nouredine Melab, Yacine Kessaci, Young Choon Lee, E-G Talbi, Albert Y Zomaya, e Daniel Tuyttens. A parallel bi-objective hybrid metaheuristic for energy-aware scheduling for cloud computing systems. *Journal of Parallel and Distributed Computing*, 71(11):1497–1508, 2011.

Kary ACS Ocaña, Daniel de Oliveira, Eduardo Ogasawara, Alberto MR Dávila, Alexandre AB Lima, e Marta Mattoso. Sciphy: A cloud-based workflow for phylogenetic analysis of drug targets in protozoan genomes. In *Advances in Bioinformatics and Computational Biology*, pages 66–70. Springer, 2011.

Kary ACS Ocana, Vitor Silva, Daniel de Oliveira, e Marta Mattoso. Data analytics in bioinformatics: Data science in practice for genomics analysis workflows. In *International Conference on e-Science*, pages 322–331, Munich, Germany, 2015. IEEE.

Eduardo Ogasawara, Daniel de Oliveira, Fernando Chirigati, Carlos Eduardo Barbosa, Renato Elias, Vanessa Braganholo, Alvaro Coutinho, e Marta Mattoso. Exploring many task computing in scientific workflows. In *Workshop on Many-Task Computing on Grids and Supercomputers*, page 2, Portland, USA, 2009. ACM.

Eduardo Ogasawara, Jonas Dias, D Oliveira, Fábio Porto, Patrick Valduriez, e Marta Mattoso. An algebraic approach for data-centric scientific workflows. *VLDB Endowment*, 4(12):1328–1339, 2011.

Daniel Cardoso Moraes de Oliveira. *Uma Abordagem de Apoio à Execução Paralela de Workflows Científicos em Nuvens de Computadores*. PhD thesis, Universidade Federal do Rio de Janeiro, 2012.

Daniel de Oliveira, Eduardo Ogasawara, Fernanda Baião, e Marta Mattoso. Scicumulus: A lightweight cloud middleware to explore many task computing paradigm in scientific workflows. In *International Conference on Cloud Computing (CLOUD)*, pages 378–385, Miami, USA, 2010. IEEE.

Daniel de Oliveira, Kary Ocana, Eduardo Ogasawara, Jonas Dias, Fernanda Baiao, e Marta Mattoso. A performance evaluation of x-ray crystallography scientific workflow using scicumulus. In *International Conference on Cloud Computing (CLOUD)*, pages 708–715, Washington, USA, 2011. IEEE.

Daniel de Oliveira, Kary ACS Ocaña, Fernanda Baião, e Marta Mattoso. A provenance-based adaptive scheduling heuristic for parallel scientific workflows in clouds. *Journal of Grid Computing*, 10(3):521–552, 2012a.

Daniel de Oliveira, Eduardo Ogasawara, Kary Ocaña, Fernanda Baião, e Marta Mattoso. An adaptive parallel execution strategy for cloud-based scientific workflows. *Concurrency and Computation: Practice and Experience*, 24(13):1531–1550, 2012b.

Suraj Pandey, Linlin Wu, Siddeswara Mayura Guru, e Rajkumar Buyya. A particle swarm optimization-based heuristic for scheduling workflow applications in cloud computing environments. In *International Conference on Advanced information networking and applications (AINA)*, pages 400–407, Perth, Australia, 2010. IEEE.

Gilberto Carvalho Pereira e Nelson FF Ebecken. Combining in situ flow cytometry and artificial neural networks for aquatic systems monitoring. *Expert Systems with Applications*, 38(8):9626–9632, 2011.

Xiao Qin e Hong Jiang. A dynamic and reliability-driven scheduling algorithm for parallel real-time jobs executing on heterogeneous clusters. *Journal of Parallel and Distributed Computing*, 65(8):885–900, 2005.

Rizos Sakellariou, Henan Zhao, Eleni Tsiakkouri, e Marios D Dikaiakos. Scheduling workflows with budget constraints. In *Integrated research in GRID computing*, pages 189–202. Springer, 2007.

SBC. Grandes desafios da computação no brasil: 2006-2016, 2006. URL `http://www.gta.ufrj.br/rebu/arquivos/SBC-Grandes.pdf`. [Web; Acessado em 23-01-2016].

Yogesh L Simmhan, Beth Plale, e Dennis Gannon. A survey of data provenance in e-science. *ACM Sigmod Record*, 34(3):31–36, 2005.

C. Soanes e A. Stevenson. Oxford dictionary of english, 2003.

Vítor Silva Sousa. Uma estratégia de execução paralela adaptável de workflows científicos. Master's thesis, Universidade Federal do Rio de Janeiro, 2014.

AKM Talukder, Michael Kirley, e Rajkumar Buyya. Multiobjective differential evolution for scheduling workflow applications on global grids. *Concurrency and Computation: Practice and Experience*, 21(13):1742–1756, 2009.

Guilherme Horta Travassos e Márcio O Barros. Contributions of in virtuo and in silico experiments for the future of empirical studies in software engineering. In *Workshop on Empirical Software Engineering the Future of Empirical Studies in Software Engineering*, pages 117–130, Roma, Italy, 2003.

Jeffrey D. Ullman. Np-complete scheduling problems. *Journal of Computer and System sciences*, 10(3):384–393, 1975.

Luis M Vaquero, Luis Rodero-Merino, Juan Caceres, e Maik Lindner. A break in the clouds: towards a cloud definition. *ACM SIGCOMM Computer Communication Review*, 39(1):50–55, 2008.

Saverio Vicario, Bachir Balech, Giacinto Donvito, Pasquale Notarangelo, e Graziano Pesole. The biovel project: Robust phylogenetic workflows running on the grid. *EMBnet. Journal*, 18(B):pp–77, 2012.

Gregor Von Laszewski, Mihael Hategan, e Deepti Kodeboyina. Java cog kit workflow. In *Workflows for e-Science*, pages 340–356. Springer, 2007.

Jianwu Wang, Daniel Crawl, e Ilkay Altintas. Kepler+ hadoop: a general architecture facilitating data-intensive applications in scientific workflow systems. In *Workshop on Workflows in Support of Large-Scale Science*, page 12, Portland, USA, 2009. ACM.

Jia Yu, Michael Kirley, e Rajkumar Buyya. Multi-objective planning for workflow execution on grids. In *International Conference on Grid Computing*, pages 10–17, Texas, USA, 2007. IEEE Computer Society.

Fan Zhang, Junwei Cao, Keqin Li, Samee U Khan, e Kai Hwang. Multi-objective scheduling of many tasks in cloud platforms. *Future Generation Computer Systems*, 37: 309–320, 2014.

Yong Zhao, Mihael Hategan, Ben Clifford, Ian Foster, Gregor Von Laszewski, Veronika Nefedova, Ioan Raicu, Tiberiu Stef-Praun, e Michael Wilde. Swift: Fast, reliable, loosely coupled parallel computation. In *IEEE Congress on Services*, pages 199–206, Salt Lake City, USA, 2007. IEEE.

Yong Zhao, Ioan Raicu, e Ian Foster. Scientific workflow systems for 21st century, new bottle or new wine? In *IEEE Congress on Services-Part I*, pages 467–471, Honolulu, USA, 2008. IEEE.