UNIVERSIDADE FEDERAL FLUMINENSE

ANDRÉ LUÍS DA COSTA NASCIMENTO

Uma estratégia de Escalonamento baseada em Aprendizado por Reforço para Workflows Científicos em Nuvens de Computadores

Niterói

2018

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

N244e Nascimento, André Luís da Costa
Uma estratégia de Escalonamento baseada em Aprendizado por Reforço para Workflows Científicos em Nuvens de Computadores : / André Luís da Costa Nascimento ; Daniel Cardoso Moraes de Oliveira, orientador ; Aline Marins Paes Carvalho, coorientadora. Niterói, 2018.
56 f. : il.

Dissertação (mestrado)-Universidade Federal Fluminense, Niterói, 2018.

DOI: http://dx.doi.org/10.22409/PGC.2018.m.10729996760

1. Escalonamento de tarefas. 2. Fluxo de trabalho. 3. Aprendizado por reforço. 4. Computação em nuvem. 5. Produção intelectual. I. Oliveira, Daniel Cardoso Moraes de , orientador. II. Carvalho, Aline Marins Paes, coorientadora. III. Universidade Federal Fluminense. Escola de Engenharia. IV. Título.

CDD -

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

UNIVERSIDADE FEDERAL FLUMINENSE

ANDRÉ LUÍS DA COSTA NASCIMENTO

Uma estratégia de Escalonamento baseada em Aprendizado por Reforço para Workflows Científicos 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: Engenharia de Sistemas e Informação

Orientador:
Daniel Cardoso Moraes de Oliveira

Co-orientador:
Aline Marins Paes Carvalho

Niterói

2018

ANDRÉ LUÍS DA COSTA NASCIMENTO

Uma estratégia de Escalonamento baseada em Aprendizado por Reforço para *Workflows* Científicos 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: Engenharia de Sistemas e Informação

Aprovado em Dezembro de 2018.

BANCA EXAMINADORA

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

Profª. D.Sc. Aline Marins Paes Carvalho, UFF

Prof. D.Sc. Eduardo Bezerra da Silva, CEFET/RJ

Profª. D.Sc. Lucia Maria de Assumpção Drummond, UFF

Niterói

2018

Dedico este trabalho aos meus pais, à minha esposa, ao meu irmão, aos meus familiares e amigos.

Agradecimentos

Primeiramente gostaria de agradecer aos meus pais, Cícero José do Nascimento e Maria Lúcia da Costa Nascimento, pelo apoio e incentivo que sempre me deram, priorizando o máximo de esforço com meus estudos.

À minha esposa, Fernanda da Silva Martins, por todo apoio, incentivo e paciência durante toda essa caminhada.

Ao meu orientador e amigo, professor Daniel de Oliveira, por cada palavra de apoio e incentivo. E por estar disponível nos horário mais adversos para que este trabalho fosse concluído.

À minha co-orientadora e amiga, professora Aline Paes, pelos ensinamentos e pela paciência em cada explicação durante toda esta trajetória.

Ao aluno de graduação Victor Olímpio pela dedicação, companheirismo e amizade.

Ao aluno de mestrado e amigo Marcello Willians Messina Ribeiro com quem tive o privilégio de conviver durante esta etapa.

Ao aluno de mestrado Thaylon Santos pela ajuda, esforço e incentivo.

Ao aluno de doutorado Vitor Silva pela ajuda e esclarecimento.

Aos membros da banca Prof. D.Sc. Eduardo Bezerra da Silva e Prof^{a}. D.Sc. Lucia Maria de Assumpção Drummond.

A todos os funcionário e professores do Instituto de Computação, com quem tive o privilégio de interagir ao longo desta etapa.

Resumo

Experimentos científicos são comumente modelados em *Workflows* Científicos. Estes *Workflows* geralmente são computacionalmente intensivos. Com isso, ambientes de processamento de alto desempenho (PAD), tais como clusters, grades e nuvem de computadores são bastante utilizados. Este último ambiente oferece a vantagem da elasticidade, que permite aumentar e/ou diminuir a quantidade de máquinas virtuais (VMs) a serem instanciados, sob demanda, sem que os cientistas necessitem adquirir infraestrutura própria. Além disso, geralmente, os *workflows* são modelados, gerenciados e executados em Sistemas de Gerência de *Workflows* Científicos (SGWfC) e muitos destes sistemas oferecem apoio a execuções na nuvem, como por exemplo o SciCumulus. Cada SGWfC possui um escalonador de atividades próprio que segue um modelo de custo bem definido. A dificuldade de se criar modelos de custos para escalonar *workflows* na nuvem é que precisamos modelar manualmente grande parte das características do ambiente, o que torna esta tarefa bastante complexa. Sem contar que eventos como migração de máquinas virtuais ou flutuações de desempenho não são facilmente modelados. Assim, esse trabalho propõe criar um escalonador que utiliza técnicas de Aprendizado por Reforço (AR - da área de aprendizado de máquina) para descobrir como melhor escalonar as ativações do *workflow*. Para tanto, foi desenvolvido uma extensão de um conhecido simulador de *workflows* denominado de *WorkflowSim*, chamada de *WorkflowSim4RL* que implementa os conceitos do aprendizado por reforço onde o aprendizado ocorre em ambiente simulado. Após obtido o plano de escalonamento, este plano é aplicado em ambiente de nuvem através da extensão do *SciCulumus*, chamada de *SciCumulus4RL*. Para a avaliação deste trabalho foi utilizado o *Montage*, que é um *workflow* científico usualmente utilizado em trabalhos relacionados a este tema. Os resultados obtidos pela execução deste *workflow* no *SciCumulus4RL* foram promissores, pois o escalonador proposto foi mais eficiente que o escalonador comumente conhecido FCFS (em inglês, *First come - First Server*, ou seja, o primeiro a chegar é o primeiro a ser servido) para ambientes com muitas máquinas virtuais.

Palavras-chave: *Workflow* científico, escalonamento, aprendizado por reforço, nuvem de computadores, simulação, Sistemas de Gerência de Workflows Científico.

Abstract

Scientific experiments are commonly modeled in Scientific Workflows. These Workflows are usually computationally intensive. Thus, high-performance processing (HPC) environments such as clusters, grids, and cloud computing are widely used. This last environment offers the advantage of elasticity, which allows to increase and/or decrease the amount of virtual machines (VMs) to be instantiated, on demand, without the scientists needing to acquire their own infrastructure. In addition, workflows are typically modeled, managed, and run on Scientific Workflows (SWfMS) systems, and many of these systems support cloud execution such as SciCumulus. Each SWfMS has its own activity scheduler that follows a well-defined cost model. The difficulty in creating cost models for scheduling workflows in the cloud is that we need to manually model most of the features of the environment, which makes this task quite complex. Not to mention that events such as virtual machine migration or performance fluctuations are not easily modeled. This work proposes to create a scheduler that uses Reinforcement Learning (RL - from the Machine Learning area) techniques to find out how best to scheduler workflow activations. For this, an extension of a well-known workflows simulator called WorkflowSim4RL was developed that implements the concepts of reinforcement learning where learning occurs in a simulated environment. After obtaining the scheduling plane, this plane is applied in a cloud environment through the extension of SciCulumus, called SciCumulus4RL. For the evaluation of this work was used the Montage, which is a scientific workflow usually used in works related to this theme. The results obtained by the execution of this workflow in SciCumulus4RL were promising, since the proposed scheduler was more efficient than the commonly known scheduler FCFS (First come, First Served) for environments with many virtual machines.

Keywords: Scientific workflow, scheduling, reinforcement learning, cloud computing, Simulation, Scientific Workflow Management Systems.

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas e Siglas

AR	:	Aprendizado por Reforço
AM	:	Aprendizado de Máquina
PAD	:	Processamento de Alto Desempenho
SGWfC	:	Sistema de Gerência de *Workflow* Científico
VM	:	Máquina Virtual
XML	:	*Extensible Markup Language*

Sumário

Capítulo 1

Introdução

Ao longo do tempo, o processo de experimentação científica se modificou bastante e diversas áreas da ciência têm se apoiado cada vez mais na computação. Como esses processos são complexos e economicamente custosos, simulações computacionais de experimentos científicos têm se tornado cada vez mais necessárias. Assim, surgiram os experimentos *insilico* [26], que são uma categoria de experimentos em que todo o processo de experimentação é baseado em simulações computacionais.

Devido à complexidade dos processos de experimentação científica e do grande volume de dados consumidos, surgiram os *workflows* científicos [8]. Esses *workflows* têm como objeto modelar de forma organizada e expor todas as etapas e suas relações de um processo científico. Para isso, o *workflow* é descrito através de um grafo, onde os vértices representam atividades de processamento de dados e arestas representam dependências entre as mesmas. Cada atividade do *workflow* representa uma etapa do processo a ser executada por um programa de computador (ou um serviço *Web*), e cada execução de uma atividade com um determinado conjunto de parâmetros é chamada de *ativação* que processa um conjunto de dados de entrada produzindo um conjunto de dados de saída [20].

Comumente, os *workflows* científicos são modelados, gerenciados e executados por Sistemas Gerenciadores de *Workflows* (SGWfC). Esses sistemas são programas complexos e robustos que permitem gerenciar o imenso volume de dados envolvidos na execução de *workflows*. Além dos SGWfCs armazenar os dados inerentes da execução do próprio *workflow*, esses sistemas também armazenam metadados chamados de proveniência [4]. Esses metadados são informações relacionadas ao experimento, tais como a definição do *workflow*, bem como os dados iniciais utilizados e os dados gerados durante a sua execução e como eles se relacionam. Dessa forma, esses dados permitem a reprodução do

experimento por outro cientista e, por esta razão, são tidos como fundamentais para que o experimento seja considerado consistente e válido [4]. Alguns SGWfCs têm como característica possibilitar sua execução em Ambientes de Processamento de Alto Desempenho (PAD) como *clusters* de computadores, grades computacionais, e mais recentemente em nuvens de computadores. As nuvens, como por exemplo a *Amazon AWS*[1] (plataforma utilizada nesta dissertação), *Google Cloud Platform*[2], *Locaweb*[3] e *Microsoft Azure*[4] têm como vantagem a elasticidade e personalização de máquinas virtuais e, a partir dessas características, é possível adequar a quantidade de máquinas virtuais (VMs) utilizadas para execução de um determinado *workflow* científico. Afinal, em ambientes de nuvem o cientista só paga pelos recursos que consome *(pay-per-use)*. Com isso, um bom SGWfC deve ter a capacidade de distribuir as execuções dos *workflows* nesses ambientes de forma eficiente a fim de explorar essas características.

Atualmente, existem diversos SGWfCs que permitem execução em nuvens de computadores, como o *Pegasus*[5] [9], o *Swift/T*[6] [31] e o *SciCumulus*[7] [21] (este último SGWfC utilizado nesta dissertação), e todos eles possuem escalonadores específicos para o ambiente de nuvem. O escalonamento de ativações é um problema de classe NP, isto é, não polinomial. Por isso, encontrar planos de escalonamentos ótimos é uma tarefa árdua e na maioria das vezes computacionalmente inviável. Neste sentido, muitas heurísticas têm sido criadas para resolver o problema de escalonamento em diferentes ambientes e modelos. No cenário atual, cada escalonador trabalha com o intuito de maximizar critérios previamente definidos como desempenho, custo financeiro, disponibilidade, etc. Ou seja, cada escalonador é definido através de um modelo que busca formalizar as características do ambiente de nuvem e distribuir as ativações nesse ambiente visando atingir seu objetivo. Entretanto, além de ser uma tarefa complexa modelar e formalizar esses ambientes, existem características que são difíceis de se prever, como é o caso de flutuações de desempenho das *VMs* e migrações de *VMs* durante a execução do experimento (*live migration*). Além disso, os escalonadores se baseiam em estimativas previamente informadas como desempenho das ativações sem as quais o escalonador trabalha de forma imprecisa e pouco eficiente. Com isso, obter tais informações sobrecarrega o trabalho do cientista como por exemplo, através de análise de histórico de execuções anteriores. E

[1] https://aws.amazon.com/pt/ec2/
[2] https://cloud.google.com/
[3] https://www.locaweb.com.br/cloud/
[4] https://azure.microsoft.com/pt-br/
[5] https://pegasus.isi.edu/
[6] http://swift-lang.org/Swift-T/
[7] https://scicumulusc2.wordpress.com/

ainda, existem situações que ele não tem como obtê-las. Dessa forma, o ideal seria que o escalonador dos SGWfCs fosse adaptativo ao ambiente ao invés da necessidade de modelar ou formalizar suas características previamente.

A solução para este problema pode ser o Aprendizado por Reforço [25] que é uma área de Aprendizado de Máquina [18] que permite essa adaptabilidade, pois baseia-se no conceito da existência de um agente capaz de aprender a ter tomadas de decisões mais eficientes em ambientes dinâmicos através de tentativas. Após cada tentativa o agente pode receber uma *recompensa* caso de resposta positiva, ou uma *punição*, em caso negativo. Este conceito satisfaz o problema já que é possível corresponder o agente ao escalonador e o ambiente a nuvem de computadores. Com isso, o escalonador não precisa conhecer previamente o ambiente, ou melhor, os desempenhos das ativações. Ele precisa ser capaz de avaliar a tentativa de escalonamento positivamente ou negativamente, o que pode ser calculado através de comparações com execuções anteriores.

O trabalho proposto nesta dissertação é a criação de um escalonador de *workflow* científico baseado em Aprendizado por Reforço em ambientes de nuvens com suporte a proveniência, usando informações de execução anterior do *workflow* como entrada para o algoritmo de escalonamento baseado em Aprendizagem por Reforço. Nesta dissertação, fazemos as seguintes contribuições. Primeiro, propomos uma extensão do *WorkflowSim*, que é um simulador de execução de *workflows* científicos, [5] chamada de *WorkflowSim4RL* para implementar o algoritmo de escalonamento proposto baseado em Aprendizado por Reforço. Dessa forma, o aprendizado ocorre em ambiente simulado, e portanto, traz vantagens ao cientista já que além do tempo de aprendizado ser menor, não o onera financeiramente, afinal esta etapa pode ser executada fora do ambiente de nuvens ou por uma única máquina na nuvem. Segundo lugar, propomos o *SciCumulus4RL*, que é uma arquitetura estendida do *SciCumulus* para executar *workflow* em ambiente de nuvem seguindo o plano de escalonamento gerado pelo *WorkflowSim4RL*. Por fim, realizamos uma avaliação experimental, baseada na implementação do *SciCumulus4RL* na *Amazon AWS* usando o *workflow Montage*[8] que é um *workflow* científico usualmente utilizado em trabalhos relacionados a este tema, além de ser amplamente conhecido.

Além deste capítulo introdutório, esta dissertação está organizada em seis capítulos. O Capítulo 2 apresenta a fundamentação teórica para a formulação do trabalho descrito por esta dissertação. Sendo assim, apresenta os conceitos: experimento científico, *workflow* científico, proveniência, *WorkflowSim*, *SciCumulus*, aprendizado por reforço, propriedade

[8]http://montage.ipac.caltech.edu/

de *Markov* e *Q-Learning*. O Capítulo 3 apresenta o *WorkflowSim4RL* que é uma extensão do *WorkflowSim* que implementa um escalonador baseado em aprendizado por reforço pra ambientes de nuvens, onde o aprendizado ocorre em ambiente simulado. E o *SciCumulus4RL* que é extensão do *SciCumulus* capaz de executar um *workflow* científico seguindo o plano de escalonamento criado pelo *WorkflowSim4RL*. O Capítulo 4 apresenta a avaliação experimental da abordagem proposta através da execução do *workflow* científico *Montage*. O Capítulo 5 apresenta os trabalhos relacionados. E por fim, o Capítulo 6 apresenta a conclusão desta dissertação ressaltando as contribuições, limitações e trabalhos futuros provenientes deste trabalho.

Capítulo 2

Fundamentação Teórica

Este capítulo aborda os conceitos que serviram como alicerce para a construção desta dissertação. Dentre eles estão: definições de experimento científico, *workflow* científicos e proveniência. Também apresenta o *WorkflowSim*, um simulador de execução de *workflows* e *SciCumulus*, que é um Sistema Gerenciador de *Worflows* (SGWfC). Além disso, são apresentados os conceitos de aprendizado de máquina, aprendizado por reforço, definição de retorno esperado, propriedade de Markov, estratégia livre de modelo comparando com a baseada em modelo e, finalmente, o *Q-Learning*, juntamente com o seu algoritmo.

2.1 Experimento Científico

É chamado de experimento científico qualquer teste executado sob condições controladas que é realizado com o objetivo de demonstrar um fato conhecido, examinar a validade de uma hipótese ou determinar a eficácia de algo que ainda não foi analisado [1]. Dessa forma, a metodologia cientifica, que é a área que estuda os métodos ou instrumentos científicos, tem como objetivo criar um conjunto de técnicas e processos para dirigir o estudo, validando o trabalho de pesquisa e/ou formulação de uma produção científica.

Assim como a ciência tem se aprimorado muito ao longo do tempo, suas metodologias também. Com isto, somado ao advento da computação, experimentos científicos passaram a se beneficiar fortemente da infraestrutura computacional existente. Dessa forma, surgiu uma nova categoria de experimento chamada de experimentos *in silico* [26]. Nesse tipo de experimento, todo o processo de experimentação é baseado em simulações computacionais. Ou seja, tanto os objetos de estudo quanto os ambientes são simulados através de modelos computacionais [11]. Assim, muitas dessas simulações são compostas pela execução de diversas aplicações científicas que possuem dependência de dados entre si, e os dados

produzidos por uma aplicação são consumidos pela próxima aplicação no fluxo [17]. Este tipo de experimento tem aumentado significativamente nos últimos anos principalmente devido a capacidade de executar experimentos em larga escala. Seja devido ao grande volume de dados a ser processado ou pela possibilidade de execução dos processos em Ambientes de Alto Desempenho (PAD), tais como *clusters*, grades computacionais e mais recentemente, nuvens de computadores. Este último como vantagem a elasticidade e a personalização das máquinas virtuais. Afinal, estes ambientes geralmente cobram pelos recursos consumidos *(pay-per-use)*, evitando o desperdício de recursos computacionais e com isso, evitando o desperdício financeiro.

2.2 *Workflows* Científicos

Workflows científicos são considerados um padrão para modelagem de experimentos científicos baseados em simulações computacionais. Estes experimentos são intensivos tanto computacionalmente quanto em volume de dados [32, 8]. Dessa forma, os *workflows* são abstrações que representam um fluxo de dados entre atividades, em que cada atividade pode representar um programa que executa operações como carregamento de dados, processamento de dados e agregação de dados.

Um *workflow* W pode ser definido formalmente através de um grafo direcionado acíclico (DAG), onde os nós $A = \{a_1,a_2, \ldots, a_n\}$ representam as atividades e as arestas Dep representam a dependência de atividades entre a atividade em A. Dado $a_i \mid (1 \leq i \leq n)$, e seja $I = \{i_1, i_2, .., i_m\}$ os dados de entrada para uma atividade a_i, então $input(a_i) \subset I$. Além disso, considere O como o conjunto de dados de saída produzidos por uma atividade a_i, então $output(a_i) \subset O$. As dependências entre duas atividades a_i e a_j é representada por $dep(a_i, a_j) \leftrightarrow \exists O_k \in input(a_j) | O_k \in output(a_i)$.

Além disso, define-se *ativação* [20] como a menor unidade de trabalho que pode ser processada em paralelo e que consome um pedaço específico de dados [13]. Considere também $Ac = \{ac_1, ac_2, \ldots, ac_k\}$ como um conjunto de ativação de *workflow* W. Cada ac_i é associado com uma atividade específica a_i que é representada por $act(ac_i) = a_i$. As ativações também apresentam dependências de dados, portanto, $input(ac_i) \in I$ e $output(ac_i) \in O$ e a dependência entre as duas ativações ac_i e ac_j pode ser representada por $dep(ac_i, ac_j) \leftrightarrow \exists r \in input(ac_j) | r \in output(ac_i) \wedge dep(act(ac_i), act(ac_j))$. A figura 2.1 mostra um exemplo de um *workflow* composto por três atividades, cada uma com duas ativações e suas dependências de dados.

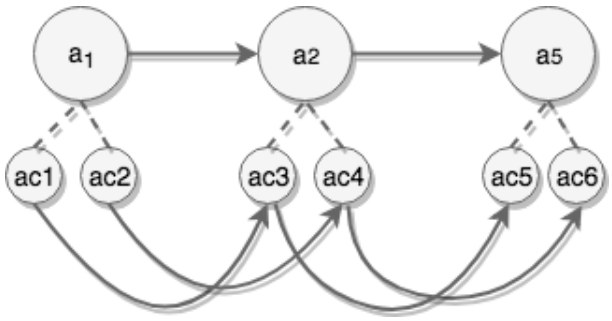

Figura 2.1: *Um exemplo de atividades de workflow e ativações*

Atualmente, os *workflow* científicos são aplicados em muitas áreas do conhecimento, como biomedicina, farmacologia, astronomia e etc [32, 19]. Estes *workflow* são, comumente, gerenciados por sistemas complexos denominados Sistemas de Gerência de *Workflow* Científicos (SGWfC), como *Pegasus* [9], *Swift/T* [31] e *SciCumulus* [21] (que foi utilizado nesta dissertação). Além de gerenciar, os SGWfC são capazes de executar, monitorar e coletar dados de proveniência [10] (que são metadados que contém informação de definição do *workflow*, dados iniciais, dados gerados a cada etapa e como estes dados se relacionam). Dessa forma, os dados de proveniência representam o histórico de execução do *workflow*. Com o advento da nuvem de computadores, alguns SGWfCs se adaptaram e passaram a atender de forma especializada o *workflow* científico através de distribuição do uso intensivo de dados e de computação [27] entre as Máquinas Virtuais (VMs) e outros recursos, como armazenamento distribuído e elasticidade. O ambiente de nuvens já demostrou ser bastante eficaz para execução de *workflows* científicos. Dessa forma, para executar um *workflow* científico na nuvem todas as ativações precisam ser programadas e executadas em paralelo em um conjunto de VMs representado por $VM = \{vm_1, vm_2, \ldots, vm_d\}$. Portanto, um SGWfC eficiente deve ser capaz de distribuir as execuções dos *workflows* nestes ambientes de forma eficaz a fim de evitar desperdícios, seja financeiro ou computacional.

Ao longo do tempo, diversos algoritmos de escalonamento já foram propostos a partir de algoritmos mais complexos que consideram a redução de desempenho [28, 16, 2], localização de dados [15], multi-nuvem [12, 14], *etc.* Em cada SGWfC é implementado algum algoritmo em um componente do escalonador. Assim, cada algoritmo de escalonamento trabalha de forma específica, a fim de maximizar critérios específicos, como custo financeiro, disponibilidade, *etc.* Com isso, nas abordagens acima mencionadas, cada algoritmo segue um modelo de custo específico que visa formalizar as características do ambiente de nuvem, com o objetivo de agendar todas as ativações em Ac para VM.

Apesar destas abordagens serem suficientes para modelar critérios do algoritmo de escalonamento, elas não consideram todas as características do ambiente de nuvem ao agendar ativações, como migrações ao vivo, flutuações de desempenho, *etc.* Na verdade, é difícil, ou às vezes inviável, modelar todas essas características, porque eles normalmente são provedores de nuvem ou aplicações dependentes [16]. Além disso, esses algoritmos são geralmente baseados em estimativas de desempenho de ativação. E estimar o desempenho das ativações em VM também é uma tarefa difícil de realizar. Sem uma previsão de desempenho precisa e considerando características como migrações de VMs e flutuações de desempenho, o escalonador pode funcionar de forma imprecisa e ineficiente. Dessa forma, o ideal seria que o escalonador dos SGWfC fosse adaptável ao ambiente, em vez da necessidade de modelar ou formalizar as características da nuvem.

2.3 Proveniência

A proveniência normalmente é utilizada apoiando os *workflows* científicos, pois eles são metadados do experimento científico W, neste caso, mais precisamente do *workflow* científico. Esses metadados são informações complementares que contêm respostas para as perguntas "como, quando, onde e por que um determinado dado foi obtido? E quem o obteve?"[4]. Ou seja, são informações relacionadas ao experimento, tais como a definição do *workflow*, bem como os dados inciais utilizados e os dados gerados durante a sua execução e como eles se relacionam. Assim, a proveniência é capaz de representar o histórico de execução do *workflow*. Além disso, ela auxilia o processo de investigação dos dados, tais como, os processos de criação e validação dos dados. Dessa forma, a proveniência permite a reprodução do experimento por outro cientista e por esta razão é tida como fundamental para que o experimento seja considerado consistente e válido [4].

Através dela é possível, por exemplo [24]:

i) Mapear o fluxo dos dados;

ii) Determinar a utilização de recursos por cada ativação;

iii) Detectar erros na geração de dados e/ou no processo;

iv) Estimar a qualidade e/ou confiabilidade dos dados baseando-se na origem dos dados;

v) A replicação ou derivação de dados;

vi) A realização consultas baseadas nos metadados de origem para a descoberta de dados.

Assim, a proveniência é amplamente utilizada, pois permite a repetibilidade (propriedade que propicia que o experimento possa ser repetido diversas vezes) e/ou reprodutibilidade do experimento científico (propriedade que possibilita a capacidade de outro cientista reproduzir o mesmo o experimento). Além disso, ela oferece um melhor entendimento sobre os experimentos científicos.

2.4 O *WorkflowSim*

Conforme mencionado anteriormente, experimentos *in silico* têm sido utilizado frequentemente no meio científico. Porém, apesar de serem experimentos basicamente computacionais, eles podem ser custosos, seja por demanda de recurso computacional, financeiro ou tempo de execução. Dessa forma, foi adotado como base para esta dissertação o simulador *WorkflowSim*, por ser amplamente conhecido com suporte a ambientes com sobrecargas provenientes de um sistema heterogêneo e sujeito a falhas, além de ser capaz de suportar escalonamento dinâmico na execução de *workflows* e suportar dependências de tarefas ou armazenamento em *cluster* [5]. Outra grande vantagem da utilização do *WorkflowSim* é a sua capacidade de simular questões como falhas e flutuações de desempenho de forma bastante consistente e se aproximando muito do ambiente real [5]. Com isso, obtém-se resultados simulados muito próximos dos resultados reais, fazendo com que ele seja uma ferramenta bastante apropriada para sua utilização nesta dissertação.

O *WorkflowSim* possui 7 componentes principais, sendo os 4 primeiros componentes responsáveis pela modelagem e organização do *workflow* na máquina local e os 3 últimos componentes responsáveis pela simulação da execução do *workflow* em ambiente virtualizado:

i) **Mapeador** - responsável por mapear o arquivo de entrada (um XML que representa um grafo acíclico direcionado) e criar os objetos em memória com a estrutura do *workflow*;

ii) **Motor de agrupamento** - componente que agrupa ativações para tornar o escalonamento mais eficiente;

iii) **Máquina de *Workflow*** - controla as dependências e informa quais ativações estão disponíveis para execução;

iv) **Escalonador Local** - decide quais ativações são despachadas para o ambiente de processamento de alto desempenho;

v) **Monitor de Falhas** - verifica a ocorrência de falhas nas máquinas virtuais;

vi) **Gerador de Falhas** - encarregado de gerar falhas na execução, simulando um ambiente de nuvem real;

vii) **Escalonador Remoto** - controla a execução das ativações nas máquinas virtuais.

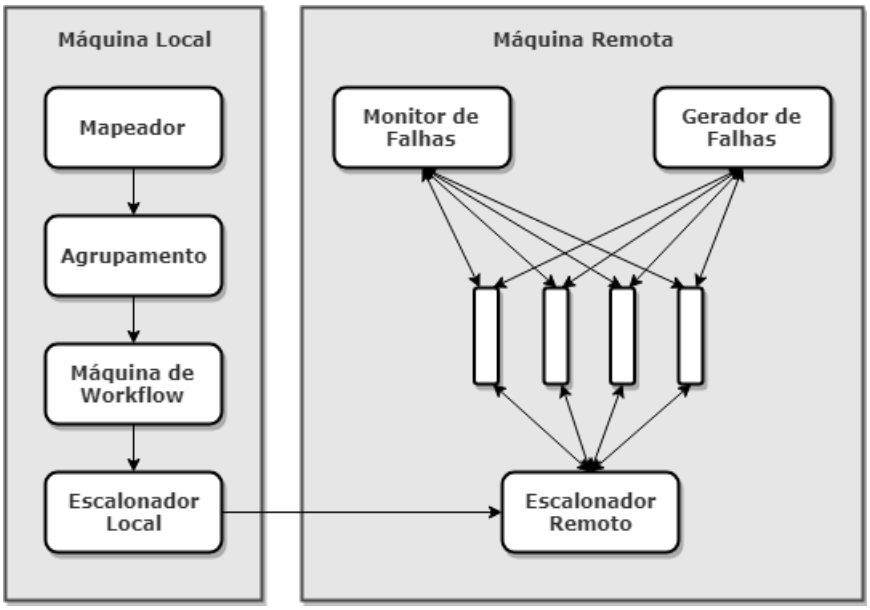

Figura 2.2: *Arquitetura do WorkflowSim [5] adaptado*

A arquitetura conceitual do *WorkflowSim* é apresentada na figura 2.2 e nela é possível visualizar o fluxo de dados gerado pela interação dos componentes do *WorkflowSim* ao executar uma simulação. Para uma melhor compreensão, a execução de um *workflow* no *WorkflowSim* ocorre da seguinte maneira: primeiramente o *workflow* precisa estar definido em um arquivo *XML* de forma que o componente mapeador seja capaz de compreender. Por ser um simulador amplamente difundido, muitos *workflows* científicos já estão disponíveis neste formato; depois, o cientista precisa definir a configuração do ambiente do experimento dentro do *WorkflowSim*, como quantidade de máquinas virtuais que serão utilizadas, quantidade de memória em cada máquina, bem como sua capacidade de processamento e etc. Esta configuração não se faz de forma parametrizada. Portanto, é necessário que o cientista defina estas características do ambiente através de programação

no *WorkflowSim*. Além disso, o cientista precisa definir qual o algoritmo de escalonamento que será utilizado para fazer a distribuição das ativações nas máquinas virtuais. O algoritmo de escalonamento padrão é o *FCFS* (*First Come - First Served* que em português significa O primeiro a chegar é o primeiro a ser servido); por último o *WorkflowSim* é executado gerando resultados baseados na especificação do *workflow* e na configuração do ambiente do experimento.

2.5 *SciCumulus* - Sistema Gerenciador de *Workflows* (SGWfC)

Os Sistemas Gerenciadores de *Worflows* (SGWfCs) são programas complexos e robustos que permitem modelar, gerenciar e executar *workflows* científicos. Além disso, existem alguns SGWfCs com suporte a dados de proveniência que permite auxiliar o trabalho do cientista em gerenciar o imenso volume de dados envolvidos na execução de *workflows* e por isso são comumente utilizados. Também existem SGWfCs que têm como particularidade possibilitar suas execuções em Ambientes de Processamento de Alto Desempenho (PAD) como *clusters* de computadores, grades computacionais, e mais recentemente em nuvens de computadores. As nuvens têm como características a elasticidade e personalização de máquinas virtuais (VMs), tornando possível adequar a quantidade de VMs utilizadas para execução de um determinado *workflow* científico, como por exemplo ocorre na plataforma *Amazon AWS*. Essas características trazem enormes vantagens ao cientista, vez que, no ambiente de nuvens o cientista só paga pelos recursos que consome *(pay-per-use)*. Com isso, um bom SGWfC deve ter a capacidade de distribuir as ativações dos *workflows* nestes ambientes de forma eficiente a fim de explorar essas características, pois se a distribuição das ativações for inadequada, além de gerar um tempo maior de execução do *worfklow*, também pode gerar um custo adicional ao cientista. O SciCumlus, por exemplo, é um SGWfC que reúne essas duas capacidades: suporte a dados de proveniência e execução de *worflows* em PAD (incluindo nuvens de computadores).

A arquitetura conceitual do *SciCumulus* é basicamente divida em 3 três camadas e cada camada é composta por um conjunto de componentes conforme se segue:

i) **Camada de cliente** - camada onde o cientista modela e configura o experimento. Os componentes dessa camada têm suporte para serem instalados em qualquer outro SGWfC existente como VisTrails, o Kepler e o Taverna;

   (a) **Carga** - responsável por mover os dados de entrada para a nuvem;

(b) **Download** - componente que traz resultados obtidos pela execução do *worklow* para a máquina do cientista;

(c) **Despacho** - responsável por gerar um arquivo de especificação/configuração (arquivo *XML*) do experimento contendo a modelagem do *workflow* e configuração do ambiente de execução;

ii) **Camada de distribuição** - controla a execução das ativações do *worflow* em nuvens;

(a) **Intermediário de execução** - faz a interface entre a camada cliente e a camada de distribuição, inicia a execução *workflow* baseado no arquivo de especificação/configuração (XML) fornecido pelo componente de despacho da camada cliente e envia mensagens sincronizadas para este mesmo componente;

(b) **Explorador de parâmetros** - componente cuja responsabilidade é manipular as combinações de parâmetros recebidos pela camada cliente para uma atividade específica do *workflow* que está sendo paralelizada;

(c) **Fragmentador de dados** - gera subconjuntos de dados fragmentados para serem distribuídos ao longo das máquinas virtuais durante a execução;

(d) **Encapsulador** - responsável por transferir os dados das ativações para as máquinas virtuais;

(e) **Escalonador** - distribui a execução das ativações entre as máquinas virtuais;

iii) **Camada de execução** - invocar os códigos executáveis do *workflow* nas máquinas virtuais;

(a) **Controlador de instância** - faz a interface entre a camada de distribuição e a camada de execução e controla o fluxo de execução na máquina virtual quando uma ativação executa mais de uma aplicação;

(b) **Configurador do ambiente** - configura ambiente de execução como variáveis de ambiente, criação de diretórios para gerenciar a execução, etc. Também cria réplicas do programa para um local específico na máquina virtual e *workspaces* (espaços de trabalho) no sistema para compartilhamento de informações sobre uma execução de uma ativação;

(c) **Executor** - executa uma aplicação específica e armazena os dados de proveniência em um repositório na nuvem;

Para uma melhor compreensão dos componentes citados juntamente como os seus relacionamentos, a Figura 2.3 demostra graficamente esta arquitetura conceitual.

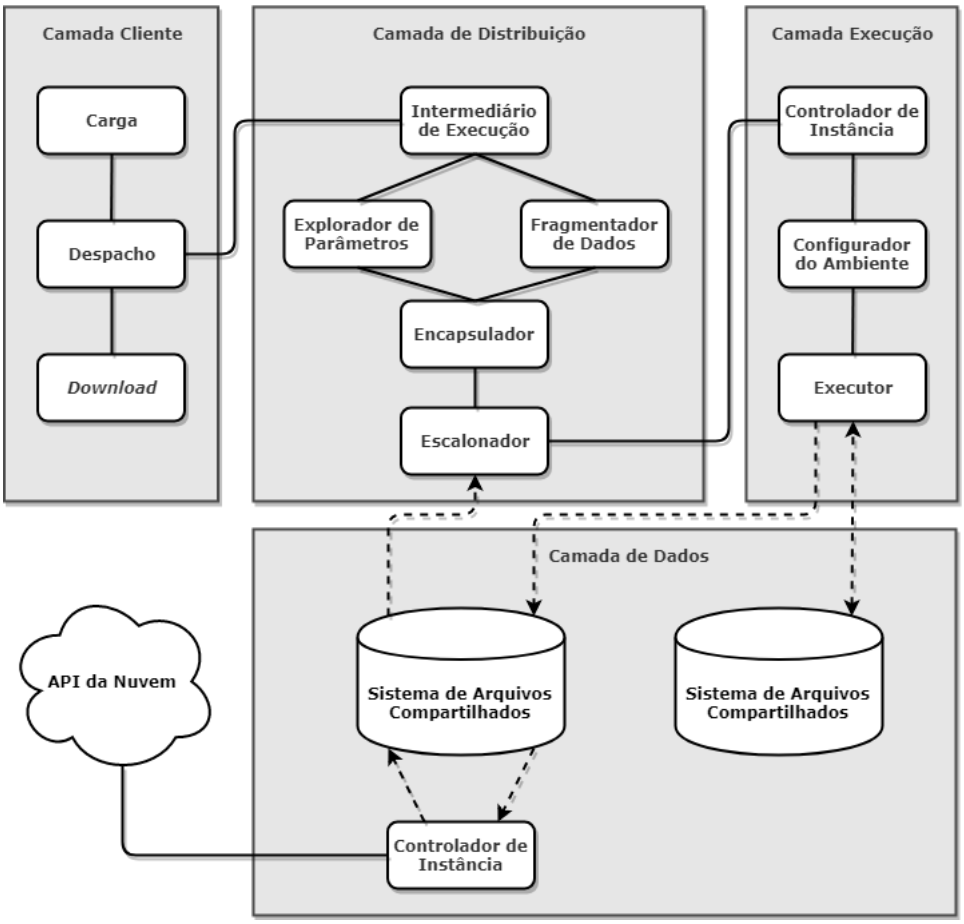

Figura 2.3: *Arquitetura conceitual do SciCumulus [7] adaptado*

Assim, para criar e executar um *worflow*, o cientista primeiro precisa definir o modelo do *workflow* através de um SGWfC, ou defini-lo diretamente no arquivo *XML* de especificação/configuração do *SciCumulus*. Caso o cientista esteja utilizando outro SGWfC, após ele modelar o *workflow* científico e configurar o ambiente informando máquinas virtuais utilizadas, configurações do banco de dados a ser armazenado os dados de proveniência, etc, o componente de despacho cria um arquivo de especificação/configuração em formato *XML* contendo todas essas informações que serão utilizadas na camada de distribuição para a execução do *workflow*. Isto permite que o cientista possa modelar e configurar o experimento em outro SGWfC (como o *VisTrails*, por exemplo) e executá-lo no *SciCumulus*. Caso o cientista opte por não usar um SGWfC diferente do *SciCumulus*, ele precisa definir o experimento diretamente no arquivo *XML* de especificação/configuração. Em seguida, o componente de intermediário de execução inicia a execução do *worflow*. Esta ação ocorre dentro da camada de distribuição, que, além disso, tem como responsabilidade

controlar a execução das ativações do *worflow* em nuvens e escalonar cada ativação para uma máquina virtual baseado na implementação definida pelo do componente escalonador no *SciCumulus*. Já a camada de execução invoca os códigos executáveis do *workflow* nas máquinas virtuais e ao término da execução do *workflow*, os dados resultantes ficam disponíveis em um repositório compartilhado, bem como os dados de proveniência. No *SciCumulus* não existe uma centralização da execução do *workflow* com distribuição das ativações dentre as máquinas virtuais no ambiente de nuvens. Cada máquina virtual do do ambiente, ou seja, cada $vm \in VM$ executa o *SciCumulus* e a sincronização das informações ocorre por meio de mensagens.

Como o escalonamento de tarefas é um problema de classe NP, isto é, não polinomial, encontrar planos de escalonamentos ótimos é uma tarefa árdua e na maioria das vezes computacionalmente inviável. Em razão disso, muitas heurísticas têm sido criadas para resolver o problema de escalonamento em diferentes ambientes e modelos. Atualmente, cada escalonador trabalha com o intuito de maximizar critérios previamente definidos como desempenho, disponibilidade, tempo de execução, etc. Assim, cada escalonador de cada SGWfC é definido especificamente para o ambiente de nuvem através de um modelo que busca formalizar as características desse ambiente e distribuir as ativações visando atingir seu objetivo. O *SciCumulus*, por exemplo, implementa seu escalonador através da formalização de definição de um modelo de custo multiobjetivo [7]. Este modelo consiste em uma série de fórmulas e para isso, o escalonador do *SciCumulus* necessita formalizar diversas caraterísticas do ambiente de nuvens. Assim, o modelo de custo definido pelo *SciCumulus* formaliza as seguintes propriedades de uma máquina virtual: quantidade de memória, capacidade de processamento juntamente com os tempos médios de execuções anteriores das ativações, índices de retardo computacional da máquina virtual, além de formalizar o tempo de execução previsto de uma ativação em uma determinada máquina virtual baseado em histórico de informação. Outras caraterísticas como a confiabilidade de cada máquina virtual que é obtida por uma função probabilística, largura de banda para transferência de dados e o custo monetário também são formalizados por este modelo. Portanto, modelar e formalizar estes ambientes é uma tarefa complexa, sem contar que ainda existem características que são difíceis de se prever, como é o caso de flutuações de desempenho das *VMs* e migrações de *VMs* durante a execução do experimento (*live migration*). Outro detalhe é que, de modo geral, os escalonadores de *workflow* de ambientes de nuvens se baseiam em estimativas previamente informadas como desempenho das ativações e sem elas o escalonador trabalha de forma imprecisa e pouco eficiente. Com isso, obter tais informações sobrecarrega o trabalho do cientista como por exemplo,

através de análise de histórico de execuções anteriores. E ainda, existem situações que ele não tem como obtê-las. Neste caso, o ideal seria que o escalonador dos SGWfCs fosse adaptativo ao ambiente em vez de modelar ou formalizar suas características previamente. Portanto, uma solução para este problema pode ser a implementação de um escalonador capaz de aprender a tomar decisões mais eficientes em ambientes dinâmicos através de tentativas e isto pode ser obtido através do aprendizado por reforço.

2.6 Aprendizado de Máquina

O Aprendizado de máquina, ou *Machine Learning*, é uma subárea da inteligência artificial (IA) [25] e segundo o Arthur Samuel é definido como o *"campo de estudos que dá aos computadores a capacidade de aprender, sem ser explicitamente programados"* [23]. Outra definição é dada por Tom Mitchell que o define como *"Um programa de computador é dito aprender com a experiência E com a relação a alguma classe de tarefas T e medida de desempenho P, se o seu desempenho medido pelo P em tarefas em T melhora com a experiência E"* [18]. Por exemplo, suponha que um robô seja capaz jogar jogo da velha (conhecido por *tic-tac-toe* em inglês). Dito isso, segundo Tom Mitchell, T representa a tarefa de jogar jogo da velha, E representa a experiência obtida ao longo da execução dessas tarefas (jogar jogo da velha) e P é a medida de desempenho que avalia as tomadas de decisão, ou seja, as jogadas durante uma partida de jogo da velha. Logo, o Aprendizado de Máquina tem como objetivo o desenvolvimento de técnicas computacionais que insiram em máquinas a capacidade de aprender, bem como a construção de sistemas capazes de adquirir alguma competência, ou conhecimento de forma automática. Estes sistemas são capazes de encontrar soluções baseadas em tomadas de decisões bem sucedidas analisando experiências acumuladas de problemas anteriores. As técnicas de Aprendizado de Máquina podem ser categorizadas em três grupos. São elas:

i) **Aprendizado Supervisionado** - técnica de aprendizado em que o dado de entrada associado ao dado saída (resultado obtido) é validado por um supervisor, isto é, nesta técnica existe um supervisor validando se o resultado obtido por uma determinada entrada está correto.

ii) **Aprendizado Não-Supervisionado** - técnica de aprendizado onde o dado de entrada associado ao dado de saída (resultado obtido) não é ser validado por um supervisor, ou seja, nesta técnica não existe um supervisor validando se o dado obtido por uma determinada entrada está correto.

iii) **Aprendizado Por Reforço (AR)** - técnica de aprendizado que é explicada na seção seguinte.

2.6.1 Aprendizado Por Reforço (AR)

Aprendizado Por Reforço (AR), do inglês *Reinforcement Learning* (RL), é a técnica de aprendizagem baseada no conceito do agente ser capaz de aprender através de tentativas de tomadas de decisão que, após cada tentativa, o agente pode receber uma *recompensa* (positiva), em caso de tomada bem sucedida, ou uma *punição* (negativa), em caso de falha. Dessa forma, o aprendizado por reforço tem como objetivo maximizar o valor numérico da soma de todas as recompensas. Diferentemente do aprendizado supervisionado e do aprendizado não-supervisionado, o aprendizado por reforço não tem como objetivo tentar encontrar um padrão ou estrutura escondido nos dados.

Por ser um paradigma que visa maximizar o valor numérico da soma de todas as recompensas, ao invés de encontrar padrões como nos outros paradigmas, é necessário que haja uma troca entre exploração e explotação. Ou seja, de modo geral, o agente deve preferir ações que já foram previamente testadas e que se provaram efetivas em produzir recompensas. Porém, se o agente sempre preferir ações baseando-se apenas em análise de ações anteriores (explotação) corre o risco de não conseguir um bom valor de recompensa final, pois, às vezes, uma punição neste momento propicia uma recompensa mais satisfatória no futuro maximizando o valor final de recompensa. Desta forma, o agente de aprendizado por reforço pode escolher ações baseado em conhecimento prévio, a fim de obter recompensas locais, ao passo que também deve explorar ações que não foram escolhidas para que melhores escolhas de ações sejam realizadas futuramente. Com isso, o agente de aprendizado por reforço pode tomar uma decisão ruim durante os processos de explotação e exploração. Para isso, o agente deve tentar uma variedade de ações e favorecer progressivamente aquelas que oferecem melhores recompensas. Na matemática este problema é recorrente e ocorre quando dado uma função $f(x)$ se procura um máximo global e fica preso em um máximo local, ou seja, para encontrar uma máximo global às vezes é necessário descer do máximo local para encontrar o máximo global.

Em aprendizado por reforço, é chamado de **agente** a parte responsável pelas tomadas de decisões e pelo aprendizado. Já, tudo o que é descrito externo ao agente, isto é, o meio ao qual o agente se encontra é chamado de **ambiente**. Consequentemente, o ambiente pode ser definido por diversos **estados**. O limite do que pertence ao agente e não ao ambiente é definido pela capacidade de controle que o agente possui e não pelo

conhecimento do agente. A interação entre o agente e o ambiente ocorre através de **ações**, que corresponde a toda e qualquer ação que o agente faz no ambiente. Já as mudanças de estados ocorrem em decorrência as reações ao ambiente às ações do agente. Toda vez, que um agente executa uma interação com o ambiente, ele recebe uma **recompensa** ou **punição** como pode ser verificada na Figura 2.4. Com isso, uma especificação completa de um ambiente, e como as recompensas são definidas, denomina-se de **tarefa**.

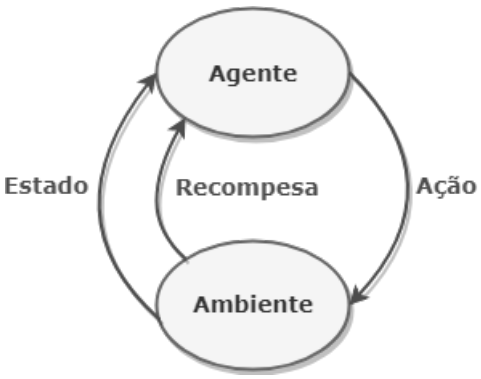

Figura 2.4: *Representação de uma interação entre o Agente e o Ambiente da definição de aprendizado por reforço adaptado [25]*

O objetivo do aprendizado por reforço é aprender uma **política** do agente, que é uma função responsável pelo mapeamento entre cada estado e a probabilidade de escolha de ações disponíveis em cada um deles. A política de comportamento do agente tem como propósito escolher ações que maximizem o valor final da soma das recompensas recebidas. Tal política é aprendida através de um processo de tentativa e erro guiado por diferentes algoritmos. Além disso, o conjunto de interações ao longo do tempo, partindo de um ambiente em **estado inicial** até atingir o **estado final**, define o conceito de **episódio**. Durante o aprendizado, ao atingir o estado final, o ambiente deve retornar ao estado inicial, independentemente do episódio anterior. Porém, este conceito só se faz presente quando existe um conjunto finito de interações que levam a um estado final do ambiente e quando um estado final é reconhecido dentro do próprio modelo. Assim, tarefas com estas características são chamadas de **tarefas episódicas**. Por outro lado, tarefas em que não possuem estado final reconhecido pelo próprio ambiente ou não possuem um conjunto finito de interações, ou seja, onde as interações ocorrem de forma contínua sem um limite definido, são chamadas de **tarefas contínuas**.

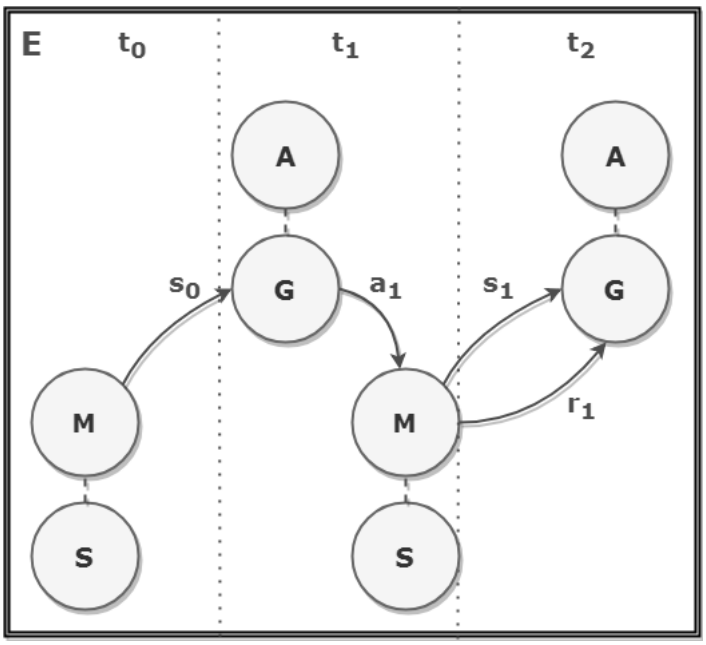

Figura 2.5: *Representação gráfica das definições de Aprendizagem por Reforço no tempo para um episódio E qualquer*

Ao expandir os conceitos representados pela Figura 2.4 (representação da iteração agente com o ambiente em um instante de tempo qualquer) é possível representar as interações do agente com o ambiente em um episódio através do tempo. Assim, o aprendizado por reforço também pode ser definido formalmente através do tempo por um grafo direcionado, onde os nós representam o agente e o ambiente e as arestas representam as interações entre eles. Para isso, considere as ações do agente G com o ambiente M podendo ser representada por A, onde A é um conjunto de ações $A = \{a_1,a_2, \ldots, a_n\}$. E S sendo um conjunto de estados definido por $S = \{s_1,s_2, \ldots, s_m\}$, que representa as reações do ambiente M as ações do agente G. Também, considere $i(s,a) \rightarrow r(a,s')$, onde $i(s,a)$ representa uma interação, sendo $s \in S$ e $a \in A$, como uma interação do agente G com o ambiente M. E $r(a,s') \in \mathbb{R}$ representa uma recompensa da interação $i(s,a)$, onde $s' \in S$ é o estado resultante da interação $i(s,a)$. Ou seja, considere um agente inserido em um ambiente, onde ele é capaz de interagir com o ambiente através de ações e o ambiente reagir através de mudança de estados. Assim, para cada interação do agente com o ambiente, o agente recebe como entrada o estado atual do ambiente e executa uma ação escolhida baseada do dilema de explotação e exploração, que consiste, respectivamente, em escolher a melhor ação dentre as ações com retorno conhecido a fim de maximizar o valor que determina a eficiência das tomadas de decisões ou explorar de forma mais ampla do espaço de soluções através de ações sem retorno conhecido. A ação executada pelo

agente propicia uma reação do ambiente com mudança do seu estado. Esta reação gera para o agente uma recompensa ou punição pela sua ação. Como cada interação entre o agente e o ambiente ocorre ao longo do tempo por meio de sequências de passos temporais, estes podem ser definidos por $T = \{t_0,t_1,t_2 \ldots, t_n\}$, onde $t \in \mathbb{R}$ e S é o conjunto de estados obtidos durante o episódio E. Todo este entendimento está representado na Figura 2.5.

Por exemplo, suponha novamente o jogo da velha. Assim, o estado inicial corresponde ao tabuleiro sem nenhuma jogada efetuada e a cada sequência de passo temporal o agente interage com o ambiente através de uma jogada. Para cada jogada, o agente recebe uma recompensa que pode contribuir para vitória ou punição, pois a jogada pode resultar em uma tomada de decisão ruim (podendo levar a derrota). A cada tomada de decisão, o ambiente reage com um novo estado do ambiente. Assim o agente terá que analisar novamente o novo estado para tomar uma nova decisão. Cada tomada de decisão ocorre sequencialmente em instantes de tempo até que o episódio seja concluído, neste caso, o fim da partida.

2.6.1.1 Retorno Esperado

Como a política, definida por $\Pi(s) : S \rightarrow A$ tem o objetivo maximizar o somatório das recompensas que recebe a longo prazo, este somatório pode ser definido formalmente através da função definida por Sutton [25]:

$$G_T = R_{t+1} + R_{t+2} + R_{t+3} + R_{t+4} + \cdots + R_T \tag{I}$$

Onde G_t é denominado **retorno esperado**, $R_t \in \mathbb{R}$ corresponde a recompensa/punição adquirida em um instante de tempo t e T é o último passo do tempo do episódio. Cada ação pode levar tempos t distintos. Por exemplo, no problema de escalonamento uma ativação complexa levará mais tempo para ser executada que uma ativação menos complexas para o mesmo recurso. Portanto, os tempos t são definidos por variáveis aleatórias. Analogamente, o tempo de término T de um episódio também não é sempre o mesmo, pois depende das tomadas de ações ao longo do episódio, portanto proporcionando tempos T diferentes.

O retorno esperado também pode ser representado pelo autômato exibido na Figura 2.6 que permite representar tarefas episódicas quanto tarefas contínuas, onde S_0 representa o estado inicial, a *caixa cinza* o estado final e $S_t \mid t \in \mathbb{N}$ todos os estados estados intermediários do episódio (estados dispostos entre o estado inicial e o estado final).

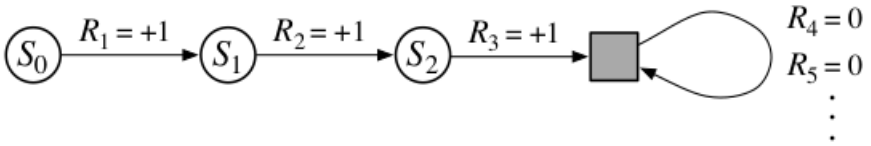

Figura 2.6: *Autômato que representa o retorno esperado. Fonte: Sutton [25]*

Através desse autômato, é possível compreender que, por exemplo, ao somar as 3 primeiras recompensas/punições de T (sendo $T = 3$) ou sobre a sequência infinita completa é obtido o retorno esperado com valor de 3.

2.6.1.2 Propriedade de Markov

Para compreender a melhor política para um determinado ambiente, o agente de aprendizado assume que o ambiente é definido pela Propriedade de Markov. Para isso, o ambiente deve torna-se um Processo de Decisão de Markov (PDM). Este processo é composto por cinco componentes. São eles:

i) Um conjunto de estados S;

ii) Um conjunto de ações A;

iii) Uma função de transição de estados (possivelmente estocástica) $T : (\mathrm{s},\ \mathrm{a}) \rightarrow s'$ que mapeia um estado para um de seus sucessores, de acordo com a ação tomada, onde s representa o estado prévio, a a ação tomada pelo agente e s' o estado imediatamente posterior a ação a;

iv) Uma função de recompensa $R : (s, a) \rightarrow \mathbb{R}$, onde s representa o estado prévio, a a ação tomada pelo agente;

v) Um fator de desconto $0 \leq \gamma \leq 1$ para avaliar o valor das ações futuras em comparação com as atuais.

Dito isso, considere um ambiente que reage à interação do agente e responda em tempo $t + 1$ a uma ação a tomada no tempo t pelo agente. Com isso, é possível definir formalmente através da Equação II que descreve a distribuição de probabilidade conjunta:

$$Pr\{S_{t+1} = s', R_{t+1} = r | S_0, A_0, R_0, \ldots, S_{t-1}, A_{t-1}, R_{t-1}, S_t, A_t, R_t\} \tag{II}$$

para todo valor de recompensa r, estado s' e todos os possíveis valores dos eventos anteriores: $S_0, A_0, R_0, \ldots, S_{t-1}, A_{t-1}, R_{t-1}, S_t, A_t, R_t$.

Se o estado do ambiente é definido pela propriedade de Markov, quando o ambiente reage à interação do agente no tempo $t+1$, a resposta depende apenas do estado s e ação a no tempo t, conforme pode ser definida formalmente pela Equação III, conhecida como equação de Markov:

$$p(s', r|s, a) = Pr\{S_{t+1} = s', R_{t+1} = r|S_t = s, A_t = a\} \quad \text{(III)}$$

Onde s' é o próximo estado do ambiente e r é a recompensa recebida pela ação a do agente ao interagir com o estado s.

2.6.1.3 Estratégias: Livre de Modelos e Baseada em Modelos

Toda interação (transição de estado) entre agente e ambiente pode ser definida por $i(s, a) \rightarrow r(a, s')$, onde uma ação a do agente em ambiente com estado s resulta em uma reação do ambiente gerando um novo estado s' e também uma recompensa/punição r. Em uma estratégia livre de modelo, não é necessário ter o conhecimento de todas as interações entre o agente e o ambiente, nem das recompensas/punições esperadas em cada estado [25]. Assim, esta estratégia depende apenas do armazenamento da interação definida através do par entre ação e estado, ou seja, $i(s, a)$ com seus respectivos valores de interação denominado Q. Estes valores são estimativas do maior retorno que o agente pode esperar após a tomada de decisão para cada estado e são obtidos ao longo do episódio através de sequência de passos temporais. Portanto, nessa estratégia não há necessidade de consultar um modelo que armazena todas as interações entre agente e ambiente de transição de estado como tentativa de obter um valor ótimo.

Em estratégia baseada em modelo, necessita-se ter o conhecimento de todas as interações (transições de estados) entre o agente e o ambiente, o que o difere da estratégia livre de modelo. Essa estratégia baseia-se em um modelo de ambiente que consiste em um modelo de transição de estado e um modelo de recompensa. O modelo de transição de estado é definido por uma árvore de decisão, em que cada nó representa um estado s do ambiente e a ligação entre os nós representa uma ação a tomada pelo agente. Assim, a árvore de decisão define todas sequências de estados obtidas através de uma sequência de ações, as quais cada estado leva a um novo estado através de uma ação do agente. O modelo de recompensa é uma associação da recompensa final obtida após percorrer uma sequência de folhas na árvore de decisão. Para utilização dessa estratégia é necessário armazenar o valor máximo da recompensa obtido por uma sequência de ações, diferentemente da estratégia livre de modelos que armazena a estimativa de maior retorno Q em

cada interação definida pelo par $i(s, a)$.

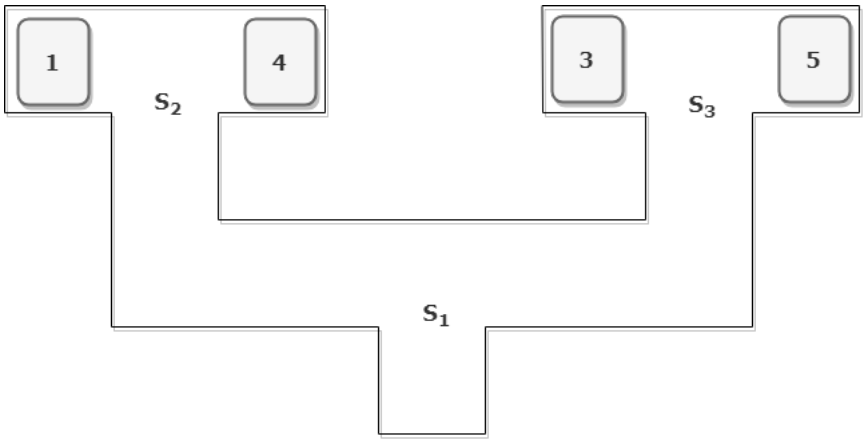

Figura 2.7: *Labirinto - exemplo de aplicação de aprendizado por reforço*

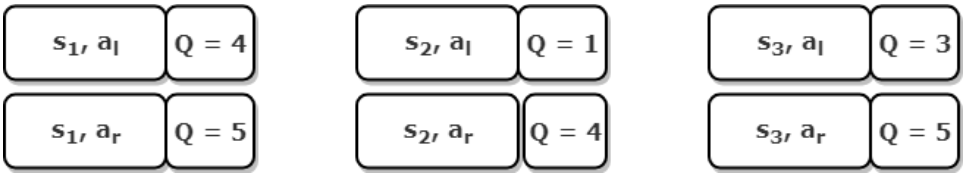

Figura 2.8: *Exemplo adaptado de Sutton [25] de pares de interações $i(a, s)$ de estratégia livre de modelo para o problema do labirinto*

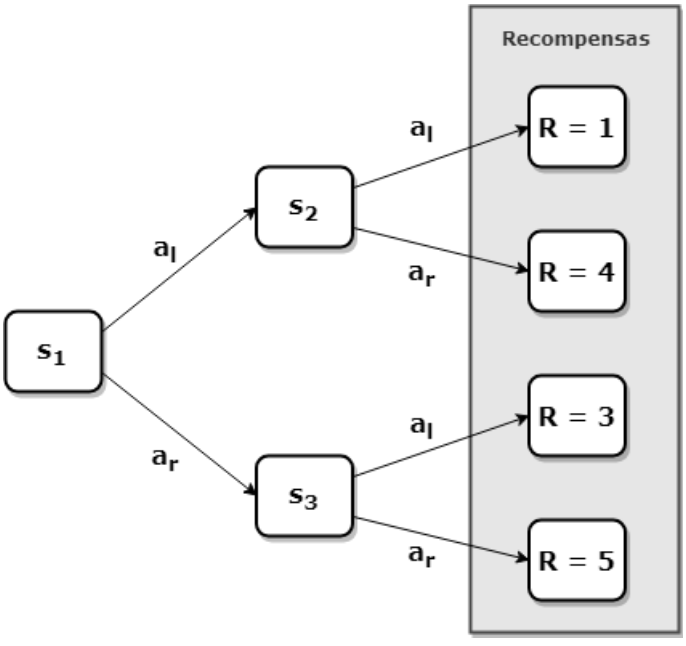

Figura 2.9: *Exemplo adaptado de Sutton [25] de árvore de decisão e modelo de recompensa de estratégia baseado em modelo para o problema do labirinto*

Para exemplificação dos conceitos, suponha que um agente robô execute as ações esquerda a_l e direita a_r em um ambiente composto por um labirinto. Dentro deste labirinto, há 4 baús de tesouros com valores 1, 4, 3 e 5 como possíveis objetivos finais a serem obtidos pelo robô. Também considere os trechos desse labirinto como os estados s_1, s_2 e s_3, conforme é exibido pela figura 2.7. Assim, a figura 2.8 representa todos os possíveis pares de interação $i(a, s)$ para uma estratégia livre de modelos. Da mesma forma, a figura 2.9 representa a árvore de decisão e o modelo de recompensa para uma estratégia baseada em modelos.

2.6.1.4 Q-Learning

Conforme descrito anteriormente, uma política ótima visa maximizar a recompensa acumulada obtida ao longo do tempo através da sequência de passos temporais. A Equação IV define formalmente esta política através da equação de recompensa:

$$\sum_{t=0}^{t=\infty} \gamma^t r_t(s, a) \tag{IV}$$

Onde t é um instante de tempo $\in \mathbb{N}$, r é a recompensa obtida em um instante t, a a ação efetuada pelo agente ao ambiente, s o estado anterior a ação a e γ é o fator de desconto do Processo de Decisão de Markov.

Além disso, como as recompensas futuras não são conhecidas inicialmente, o algoritmo *Q-Learning* tem como objetivo intrínseco, através de uma abordagem livre de modelo, aproximar o valor retornado pela função de valoração de ações denominada de Q, que mapeia um estado s e uma ação a para a recompensa esperada r. Ou seja, dado um instante de tempo t com estado s, executa uma ação a e obtenha uma recompensa r. A Equação V mostra como o valor de Q é calculado e atualizado.

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \times (r(s_t, a_t) + \gamma^t \times \max_a Q(s_{t+1}, a) - Q(s_t, a_t)) \tag{V}$$

Onde $s_t \in S$, $a_t \in A$, r é a recompensa recebida após a execução de a_t em um estado s_t no tempo t. α é a taxa de aprendizado, definindo o peso do valor atual de Q os valores futuros de Q. γ é o fator de desconto do Processo de Decisão de $Markov$. E ϵ é o dilema entre exploração/exploração é enfrentado pelo *Q-learning*, em que cada interação tem probabilidade ϵ de tomar a melhor ação baseada no valor obtido pela função Q. Caso contrário, uma ação é escolhida aleatoriamente dentre as ações possíveis.

O algoritmo *Q-Learning* é capaz de aprender uma política entre estados e ações,

usando definições que não estão atreladas a modelagem do problema, e sim, baseando-se somente na maximização do valor de Q, analisando o estado prévio e uma ação com parâmetros de entrada e o retorno sendo uma recompensa futura esperada por mapeamento pelo estado prévio, uma ação e estado resultante. Observe também, que tanto a recompensa atual $r(s_t, a_t)$, quanto a recompensa estimada futura $\max_a Q(s_{t+1}, a)$ são levados em conta ao calcular o valor da função Q. Conforme pode ser visto no Algoritmo 1, o processo de estimação do valor da função Q se repete para um certo número de episódios.

Algoritmo 1 Q-Learning

Entrada: γ, α, ϵ

Inicialize $Q(s, a) \forall s, a \in S, a \in A$, de forma arbitrária
Para cada episódio e **faça**
    $t \leftarrow 0$
    escolha um estado inicial aleatório
    Enquanto não atingir o objetivo **faça**
        $s_t \leftarrow$ o estado atual
        escolhe um valor aleatório p
        Se $p <=$ probabilidade ϵ **então**
            escolhe a melhor ação a para s_t de acordo com $Q(s, a)$
        else
            escolha uma ação aleatória a
        execute a no ambiente
        receba a recompensa $r(s, a)$
        $s_{t+1} \leftarrow$ o próximo estado
        $\delta \leftarrow (r(s_t, a_t) + \gamma^t \times \max_a Q(s_{t+1}, a) - Q(s_t, a_t))$
        $Q(s, a) \leftarrow Q(s_t, a_t) + \alpha\delta$
        $t \leftarrow t + 1$

Capítulo 3

Abordagem Proposta

De acordo com o que foi apresentado no Capítulo 2, experimentos *insilico* têm sido frequentemente utilizado no meio científico e devido a sua complexidade são definidos através de *workflows* científicos. Estes *workflows* são computacionalmente intensivos e, por isso, costumam ser modelados, gerenciados e executados por Sistemas Gerenciadores de *Workflows* (SGWfC). Como os *workflows* demandam grande capacidade de processamento e envolvem um grande volume de dados para executá-los, alguns SGWfCs permitem que suas execuções ocorram em ambientes de nuvens. Esses ambientes têm como característica a elasticidade, que permite aumentar e diminuir quantidade de máquinas virtuais a serem instanciados. Porém, distribuir as ativações do *workflow* entre as máquinas virtuais na nuvem é um problema comumente conhecido como escalonamento de tarefas. Esse problema é de classe NP, ou seja, não polinomial e, com isso, encontrar planos de escalonamentos ótimos é uma tarefa árdua e na maioria das vezes computacionalmente inviável. Apesar disso, um bom SGWfC deve ser capaz de distribuir as execuções do *workflow* científico nestes ambientes de forma eficiente, evitando o desperdício de recursos computacionais, e, assim, evitar os desperdícios financeiro e de tempo. A solução para este problema pode ser através dos conceitos de aprendizado por reforço que são descristos no Capítulo 2 na seção 2.6.1, os quais baseiam-se na existência de um agente capaz de aprender a tomar decisões mais eficientes em ambientes através de tentativas. Com isso, a abordagem proposta nesta dissertação é a criação de um escalonador que utilize técnicas de aprendizado por reforço para descobrir como melhor escalonar as ativações do *workflow*. Porém, como os *workflows* demandam muito recurso computacional e o aprendizado ocorre através de repetições de episódios, fez-se necessário que a etapa de aprendizado ocorresse em ambiente simulado. Assim, após realizado o aprendizado em um ambiente simulado, é gerado um plano de escalonamento seguido por um SGWfC em

um ambiente de nuvens. Para tanto, foi necessário criar as seguintes extensões: *WorkflowSim4RL* - estendido a partir do *WorkflowSim*); e o *SciCumlus4RL* - estendido a partir do *SciCumulus*.

Este capítulo apresenta a extensão *WorkfllowSim4RL* ressaltando sua modelagem conceitual, o processo de decisão de Markov, sua equação de desempenho/eficiência e a implementação do seu algoritmo de escalonamento *RFLSchedulingAlgorithm*. Também apresenta a extensão *SciCumulus4RL* que permite executar o escalonamento baseado em aprendizado por reforço das ativações de *workflows* científicos em ambientes de nuvem.

3.1 *WorkflowSim4RL*: Extensão do *WorkflowSim*

A proposta do *WorkflowSim4RL* é criar um escalonador que utilize os conceitos de aprendizado por reforço explicados no Capítulo 2 na Seção 2.6.1. Para isso, foram estendidos os componentes: o **escalonador local** e o **escalonador remoto**, por serem os componentes responsáveis pela decisão de alocação das ativações nas máquinas virtuais, e, portanto, são os componentes necessários para criar um escalonador baseado em aprendizado por reforço; e o **mapeador** cuja equação é mapear elementos do *workflow* de entrada (um XML) para serem simulados no *WorkflowSim* como, por exemplo, as ativações, além de ser o componente responsável por conter as classes *task* (tarefa) e *job* (trabalho), que modelam ativações, e as classes que modelam as máquinas virtuais. Com isso, estas classes tiveram que ser alteradas para armazenar informações relevantes do aprendizado como por exemplo o valor Q definido no par de interação $i(s, a)$ explicado no Capítulo 2. O detalhamento do cálculo do valor Q é explicado ao longo desta seção. Para melhor compreender a extensão *WorkflowSim4RL*, a Figura 3.1 apresenta a arquitetura original do *WorkflowSim* com todos os seus principais componentes, sendo que os componentes em destaque representam as extensões propostas pelo *WorkflowSim4RL* por esta dissertação.

Logo, esta seção tem como objetivo exibir a abordagem proposta baseando-se nos conceitos de aprendizado por reforço através de uma modelagem conceitual. Diante disso, as subseções sequentes apresentam como o ambiente foi definido no formato de um Processo de Decisão de Markov e como foram definidas as funções para punição e recompensa, e a maneira como foi obtido o valor da equação de avaliação Q do *Q-Learning*. Por fim, na última subseção é demonstrado como estes conceitos foram aplicados dentro do *WorkflowSim* resultando no *WorkflowSim4RL*.

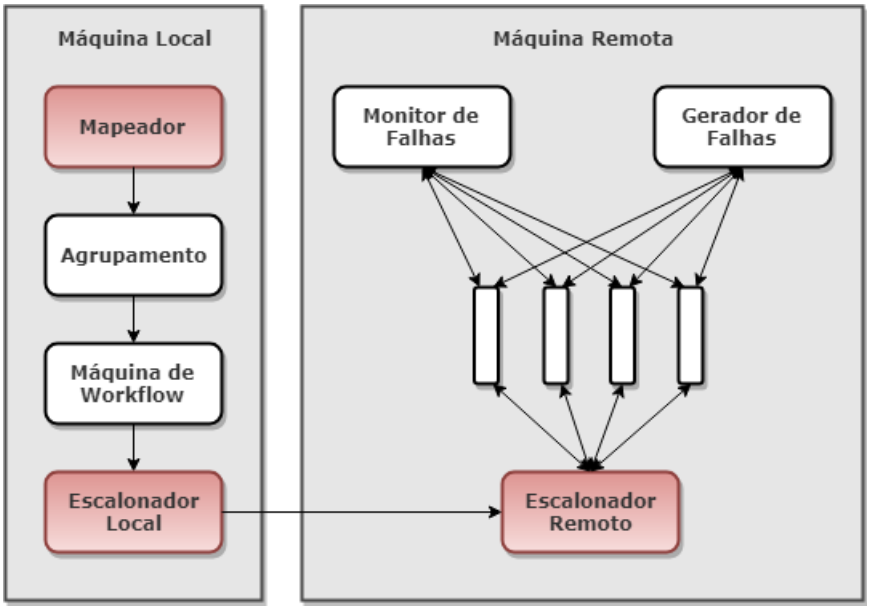

Figura 3.1: *Arquitetura do WorkflowSim [5] adaptado com as extensões destacadas*

3.1.1 Processo de Decisão de Markov

Um ambiente de aprendizado por reforço que satisfaz a propriedade Markov, explicado na Seção 2.6.1.2, é chamado de Processo de Decisão de Markov. Para tanto, é necessário definir os possíveis estados do ambiente, todas as ações e o modelo de transição. Dessa forma, admite-se que o ambiente é composto pelas máquinas virtuais disponíveis e as ativações do $workflow$ e o agente é o escalonador. Com isso, a política aprendida ao longo de um episódio define um plano de escalonamento. Contudo, o aprendizado não precisa ocorrer em um único episódio, ele pode ocorrer em uma sequência de episódios, sendo que cada episódio define um novo plano de escalonamento levando em consideração o aprendizado obtido nos episódios anteriores. Para definir formalmente este problema, considere um *workflow* W formado por um conjunto de ativações $Ac = \{ac_1, ac_2, \ldots, ac_k\}$ e um conjunto de máquinas virtuais $VM = \{vm_1, vm_2, \ldots, vm_n\}$. Dessa forma, considere que uma máquina virtual vm_z pode assumir um estado $s_{vm} \in \mathcal{SVM} = \{idle, busy\}$. Como a execução das ativações depende que elas sejam alocadas em máquinas virtuais, então considere que cada ativação pode estar em um estado $s_{ac} \in \mathcal{SAC} = \{ready, locked, running, succesfully\ finished, finished\ with\ a\ failure\}$. Após uma execução com sucesso de uma ativação s_{ac} em uma máquina virtual vm_k, o estado dessa ativação é $s_{ac} = successfully\ finished$. Entretanto, pode ocorrer de uma ativação terminar com falha devido a um problema de hardware, como problema de transferência de dados pela rede, ou outro problema qualquer. Quando isso ocorre,

o estado dessa ativação é $s_{ac} =$ *finished with a failure*. Uma ativação que está sendo executada em alguma máquina virtual tem seu estado $s_{ac} =$ *running*. Já uma ativação que está sendo impedida de ser executada por haver dependência entre ativações tem seu estado $s_{ac} =$ *locked*, ou seja, isso ocorre quando uma a ativação ac_i está esperando por outra ativação ac_j concluir de acordo com $dep(ac_i, ac_j) \leftrightarrow \exists r \in input(ac_j) | r \in output(ac_i) \wedge dep(act(ac_i), act(ac_j))$. Por último, o *workflow* W pode estar em um estado $s_w \in \mathcal{SW} = \{successfully\ finished, finished\ with\ failure, available, unavailable\}$. Estes estados são definidos com base nos estados de todas ativações que compõem o *workflow* W, e portanto, cada estado é definido conforme se segue:

- $s_w = successfully\ finished$ se $\forall\ s_{ac} = successfully\ finished$
- $s_w = finished\ with\ failure$ se $\exists\ s_{ac} = finished\ with\ failure$ e $\nexists\ s_{ac} = ready$ e $\nexists\ s_{ac} = locked$ e $\nexists\ s_{ac} = running$
- $s_w = available$ se $\exists\ s_{ac} = ready$
- $s_w = unavailable$ se $\nexists\ s_{ac} = ready$

Portanto, ao final da execução do *workflow*, se todas ativações estiverem com o estado *successfully finished* significa que o *workflow* foi executado com sucesso. Já, se houver uma única ativação com o estado *finished with failure*, significa que houve alguma falha na execução do *workflow* e com isso os dados obtidos não são válidos. Ambos estados, *successfully finished* e *finished with failure*, são estados terminais, ou seja, todas as ativações foram concluídas e a execução do *workflow* chegou ao fim. Diferentemente dos estados *available* e *unavailable* que suas representações só possuem significados durante a execução do *workflow*.

Nessa abordagem há somente dois tipos de ações permitidas: a alocação de uma ativação ac_x para uma VM vm_y definido por $alloc(ac_x,\ vm_y)$; ou fazer nada que é definido por *do nothing*. Observe, no entanto, que durante a execução pode-se ter ativações em estado $s_{ac} = ready$, ou seja prontas para serem alocadas. Porém, se não houver máquinas em estados $s_{vm} = idle$, não haverá alocação de uma ativação, e por isso, nesta situação a ação adotada é *do nothing*. Analogamente, em um dado momento pode-se ter máquinas em estados $s_{vm} = idle$, porém se não houver uma ativação ac_k em estado $s_{ac} = ready$, também a ação será *do nothing*.

A função de transição $T : S \times A \rightarrow S$ define as possíveis transições de estados do *workflow* W. O *workflow* começa com o estado $s_w = available$. Neste momento, a única

Máquina	Ativações	Workflow	Ação	Próximo Estado Workflow
idle	$\exists$ *ready*	available	$alloc(ac_i,\ vm_j)$	available
idle	$\exists$ *ready*	available	$alloc(ac_i,\ vm_j)$	unavailable
idle	$\nexists$ *ready*	unavailable	*do nothing*	available
idle	$\nexists$ *ready*	unavailable	*do nothing*	unavailable
idle	$\nexists$ *ready*	unavailable	*do nothing*	finished-with-success
idle	$\nexists$ *ready*	unavailable	*do nothing*	finished-with-failure
running	$\exists$ *ready*	available	*do nothing*	available
running	$\nexists$ *ready*	unavailable	*do nothing*	available
running	$\nexists$ *ready*	unavailable	*do nothing*	unavailable
running	$\nexists$ *ready*	unavailable	*do nothing*	finished-with-success
running	$\nexists$ *ready*	unavailable	*do nothing*	finished-with-failure

Tabela 3.1: Tabela de Transição de Estados do *workflow*

ação possível para esse estado é uma ação de alocação $alloc(ac_x,\ vm_y)$. Depois disso, W pode permanecer com o estado $s_w = available$ se $\exists\ s_{ac} = ready$, ou seja, caso ainda haja mais ativações para serem executadas. Ou W pode adquirir o estado $s_w = unavailable$ se $\nexists\ s_{ac} = ready$, isto é, caso não haja mais ativações com estado $s_{ac} = ready$. Com isso, essa função de transição não é determinística, pois uma mesma ação pode provocar estados diferentes. Já com o estado em $s_w = unavailable$, a única ação permitida é *do nothing*. Alterações no ambiente podem fazer com que o *workflow* W transite para outros estados. No caso de não haver alterações no ambiente, ou seja, nenhuma das ativações em execução foi concluída, *workflow* permanecerá com o estado $s_w = unavailable$. Caso ocorra o término de pelo menos uma ativação que permita que esta assuma o estado $s_{ac} = ready$, então o *workflow* W irá adquirir o estado $s_w = available$. Finalmente, se todas as ativações tiverem terminado e não houver mais ativações para executar, então *workflow* irá para um dos dois estados terminais. Essas transições de estados são apresentadas pela tabela 3.1.

3.1.2 Função de Desempenho/Eficiência

A função de desempenho de uma máquina virtual definida nesta dissertação foi inspirada na função que tinha como propósito servir ambientes de grade [6]. Consequentemente, ela se tornou um ponto de partida, já que a proposta desta dissertação é criar um escalonador baseado em aprendizado por reforço para ambientes de nuvens. Assim, para avaliar a qualidade de uma alocação de uma ativação ac_i em uma máquina virtual vm_j definido por $alloc(ac_i,\ vm_j)$, primeiro é preciso analisar o desempenho de uma vm_j e compará-la com o desempenho geral do *workflow*. A equação I define P_{ij} que representa

numericamente a eficiência de uma máquina virtual baseada em sua capacidade de executar uma ativação em relação ao tempo que permaneceu em fila aguardando a execução. Para isso, considere i o índice de uma ativação ac_i, j o índice de uma máquina virtual, te_i como o tempo de execução de uma ativação ac_i em uma máquina virtual vm_j e tf_i como o tempo de fila da ativação ac_i para ser executada. Considere também $tt_i = te_i + tf_i$, onde te_i é o tempo total de execução da ativação ac_i. Além disso, considere que μ representa a ponderação da relação tempo de execução com o tempo de fila.

$$P_{ij} = te_i * \mu \ + (1 - \mu) * (tf_i) \tag{I}$$

Analogamente, a equação II representa o índice de desempenho global de uma máquina virtual vm_j, ou melhor, corresponde a média dos desempenhos de todas as ativações executadas pela máquina virtual vm_j. Assim, considere $\overline{te}$ a média dos tempos de todas as ativações ac_i que foram executadas em vm_j, e $\overline{tf}$ a média dos tempos de espera na fila de todas as ativações ac_i que foram executadas em vm_j e $\overline{P_{ij}}$ é o índice de desempenho global de máquina virtual vm_j.

$$\overline{P_{ij}} = \overline{te_i} * \mu + (1 - \mu) * (\overline{tf_i}) \tag{II}$$

Já a equação III calcula o desempenho global $\overline{Pw}$ do *workflow* levando em conta todas as máquinas virtuais, onde $\overline{te}$ e $\overline{tf}$ são os tempos médios de execução e tempo de fila, respectivamente, considerando todas as ativações $ac_i \in Ac$. Assim, a equação III define desempenho global $\overline{Pw}$ do *workflow* da seguinte forma:

$$\overline{Pw} = \overline{te} * \mu + (1 - \mu) * (\overline{tf}) \tag{III}$$

A equação IV define a função de recompensa/punição parcial pela ação $alloc(ac_i,\ vm_j)$. Ela é obtida através comparação do desempenho médio $\overline{P_{ij}}$ de uma máquina virtual vm_j com o índice de desempenho global médio $\overline{Pw}$ do *workflow*. Efetuando este cálculo e calculando o desvio padrão $stdv$ de $\overline{P_{ij}}$ obtém-se:

$$r_i = \begin{cases} 1, & seP_{ij} < \overline{Pw} - stdv \\ -1, & seP_{ij} > \overline{Pw} + stdv \end{cases} \tag{IV}$$

Para calcular a recompensa/punição r^t, que atualiza o valor de Q em um instante de

tempo t, é definida a equação V:

$$r^t = r^{t-1} + \rho \times (r_i - r^{t-1}) \quad \text{(V)}$$

Onde r^{t-1} é a recompensa anterior e ρ é um parâmetro usado para medir a relevância da recompensa parcial em comparação à recompensa anterior. Assim, esta equação também fornece a intuição de que a decisão de escalonamento está melhorando a eficiência do *workflow*.

Finalmente, a atualização da eficiência de uma máquina virtual no tempo é dada pela equação VI:

$$nrle_i = rle_i + \alpha * (r_i - rle_i) \quad \text{(VI)}$$

Onde $nrle$ representa o valor da nova eficiência da máquina virtual vm_j dada a ação $alloc(ac_i,\ vm_j)$, r_i é a recompensa/punição parcial dessa mesma máquina pela ação $alloc(ac_i,\ vm_j)$, α representa a velocidade de aprendizagem definida pelo *Q-Learning* no Capítulo 2 na Seção 2.6.1.4 e rle_i é o valor atual da eficiência da máquina virtual vm_j dada a ação ac_i. Como o parâmetro rle_i deve possuir algum valor inicial para que seja calculada a nova eficiência de uma máquina virtual $nrle_i$, então nesta abordagem ele foi iniciado com um valor aleatório. O resultado obtido pela função de eficiência corresponde a um valor numérico que representa quão eficiente foi a alocação de uma determinada máquina virtual para a execução de uma ativação, e por isso, ele foi utilizado como recompensa/punição na função do *Q-Learning*. Assim, o escalonador é punido toda vez que faz uma alocação ineficiente de uma ativação a uma máquina virtual e recompensado quando faz boas alocações.

3.1.3 Implementação

Na seção anterior foi descrito que para calcular a função de desempenho de uma máquina virtual é levado em conta o tempo de execução da ativação nesta máquina virtual, bem como o tempo de fila de espera para ser executada. Porém, o valor de tempo de execução de uma ativação em uma máquina virtual só é obtido após a alocação da mesma. Portanto, primeiramente o *WorkflowSim4RL* faz com que cada ativação seja executada em cada máquina virtual, ou seja, o cartesiano de ativações *versus* máquinas virtuais. Com isso, obtêm-se todos os tempos de execução de todas as ativações em todas as máquinas virtuais. Após esta etapa, essas informações são gravadas (através de arquivo) permitindo que sejam utilizadas durante os episódios. Este processo ocorre somente antes do primeiro episódio.

Conforme explicado anteriormente, o *Q-Learning* trabalha com sequência de episódios. Assim, cada execução completa do $workflow$ representa um episódio para o algoritmo *Q-Learning*. Para fazer a interligação entre os episódios foi necessário persistir (também em arquivo) todas as informações relevantes de aprendizagem e análise para serem utilizadas nos episódios subsequentes, como a ação de alocação de ativação em uma máquina virtual, ou seja, a ação que representa o par ativação e máquina virtual definido por $alloc(ac_x,\ vm_y)$, juntamente com o valor obtido através da função de avaliação Q do *Q-Learning* e o estado do *workflow* após a alocação. Assim, no início de cada execução do *workflow* todas as informações persistidas referente ao episódio anterior são carregadas permitindo a progressão do aprendizado. Logo, para adequar esta necessidade foi imprescindível alterar modelos do componente denominado mapeador, com o intuito de armazenar os valores obtidos da função de avaliação Q do *Q-Learning* para cada ação de alocação $alloc(ac_x,\ vm_y)$, a fim de serem utilizados tanto em tempo de execução, quanto posteriormente, isto é, nos próximos episódios. Além disso, ao final do episódio é persistido o plano de escalonamento gerado, juntamente com o tempo do aprendizado (tempo de execução da simulação) e o tempo simulado da execução do *workflow*, para serem usados como métricas de avaliação no Capítulo 4.

Para o desenvolvimento do escalonador *RFLScheduling*, que é baseado em aprendizado por reforço, foi necessário implementar os conceitos mencionados nas seções anteriores deste capítulo, além do próprio *Q-Learning*. Para o seu funcionamento, o escalonador deve receber uma lista de ativações disponíveis para execução, porém ainda não escalonada, juntamente com a lista de máquinas virtuais disponíveis. O próprio *WorflowSim* se encarrega de filtrar as ativações que estão impedidas de serem executadas por questões de dependência. A tabela de avaliação Q utilizada pelo *RFLScheduling* é representada por uma matriz contendo todos os valores de Q para cada ação de alocação entre uma ativação e uma máquina virtual seguindo a definição de uma estratégia livre de modelos descrito no Capítulo 2. A seleção da máquina se dá de duas formas: caso esteja em fase de explotação, a escolha é determinada com base no maior valor da função de avaliação Q do algoritmo *Q-Learning* dentre as máquinas virtuais ociosas, isto é, com seu estado em *idle*; por outro lado, caso esteja em fase de exploração, a escolha é feita de forma aleatória com base na probabilidade ϵ. Nessa fase a escolha da máquina virtual dentre as máquinas virtuais ociosas também ocorre de forma aleatória. Depois disso, é efetuada a alocação da ativação corrente na máquina virtual escolhida. Logo após, ocorre a obtenção dos tempos de execução e de espera em fila dessa ativação para calcular o índice de eficiência P_{ij} para alocação da ativação na máquina virtual escolhida. Em seguida, calcula-se

o índice de desempenho global $overline P_{ij}$ da máquina virtual para todas as ativações executadas nesta máquina e o índice de desempenho do *workflow* $\overline{Pw}$. Posteriormente, é calculada a recompensa/punição parcial r_i, a recompensa r^t e o valor da eficiência da máquina virtual $nrle_i$. Por conseguinte, calcula atualiza-se $Q(s, a)$ baseado na função Q do *Q-Learning*. Por fim, atualiza do estado do *workflow*. Ao término do escalonamento de todas as atividades, o simulador executa toda a simulação. Ao fim deste processo, os dados obtidos do episódio são gravados para que possam ser utilizados no episódio seguinte, assim contribuindo continuamente com o aprendizado até atingir o número de episódios desejado. Dessa forma, o algoritmo 2 descreve o passo-a-passo proposto pelo escalonador *RFLScheduling* para obtenção de um plano de escalonamento através de diversos episódios sequenciais.

Algoritmo 2 RFLSchedulingAlgorithm

Entrada: $S, A, T, \gamma, \alpha, \epsilon, \mu, \rho, maxIter, Ac, VMs$

Saída: Uma matrix $Q : S \times A$ representando o plano de escalonamento

Inicializa $Q(s, a) \forall s, a, s \in S, a \in A$ aleatoriamente
$iter \leftarrow 0$
Enquanto $iter < maxIter$ **faça**
$t \leftarrow 0$
$s \leftarrow available$
$t \leftarrow 1$
$r^t \leftarrow 0$
Enquanto $s \neq successfully\ finished \wedge s \neq finished\ with\ failure$ **faça**
Se $s = available$ **então**
Com a probabilidade ϵ escolha a melhor ação a_{ij} para s de acordo com $Q(s, a)$; ou escolha uma ação a_{ij} aleatoriamente
Aloque $ac_i \in Ac$ para $vm_j \in VMs$
Calcule $P_{ij}(\mu, ac_i, vm_j)$ de acordo com a equação I
Calcule $\overline{P_{ij}}(\mu, Ac, VMs)$ de acordo com a equação II
Calcule $\overline{Pw}(\mu, Ac, VMs)$ de acordo com a equação III
Calcule $r_i(\overline{P_{ij}}, \overline{Pw})$ de acordo com a equação IV
Calcule r^t de acordo com a equação V
$\delta \leftarrow (r^t + \gamma^t \times \max'_a Q(s', a') - Q(s, a))$, onde o valor de $Q(s, a)$ corresponde a rle
$Q(s, a) \leftarrow Q(s, a) + \alpha \times \delta$, onde o novo valor de $Q(s, a)$ corresponde a $nrle$ de acordo com a equação VI
$t \leftarrow t + 1$
else
$a \leftarrow do\ nothing$
$s \leftarrow$ o novo estado s'
$iter \leftarrow iter + 1$

3.2 *SciCumulus4RL*: Extensão do *SciCumulus*

A proposta do *SciCumulus4RL* é criar um escalonador para ambientes de nuvens capaz de seguir o plano de escalonamento gerado pelo *WorkflowSim4RL* baseado em aprendizado por reforço explicado no Capítulo 2 na seção 2.6.1. Para isso, foram estendidos dois componentes que permitissem esta adaptação: o **escalonador**, por ser o componente responsável pela decisão de alocação das ativações nas máquinas virtuais na nuvem, e portanto, o componente necessário para executar o escalonamento obtido pelo *WorkflowSim4RL*; e o **despacho**, que é componente responsável por mapear o arquivo de configuração dos *workflows* científicos, com isso, permitir que o *workflows* científico seja executado pelo novo escalonador.

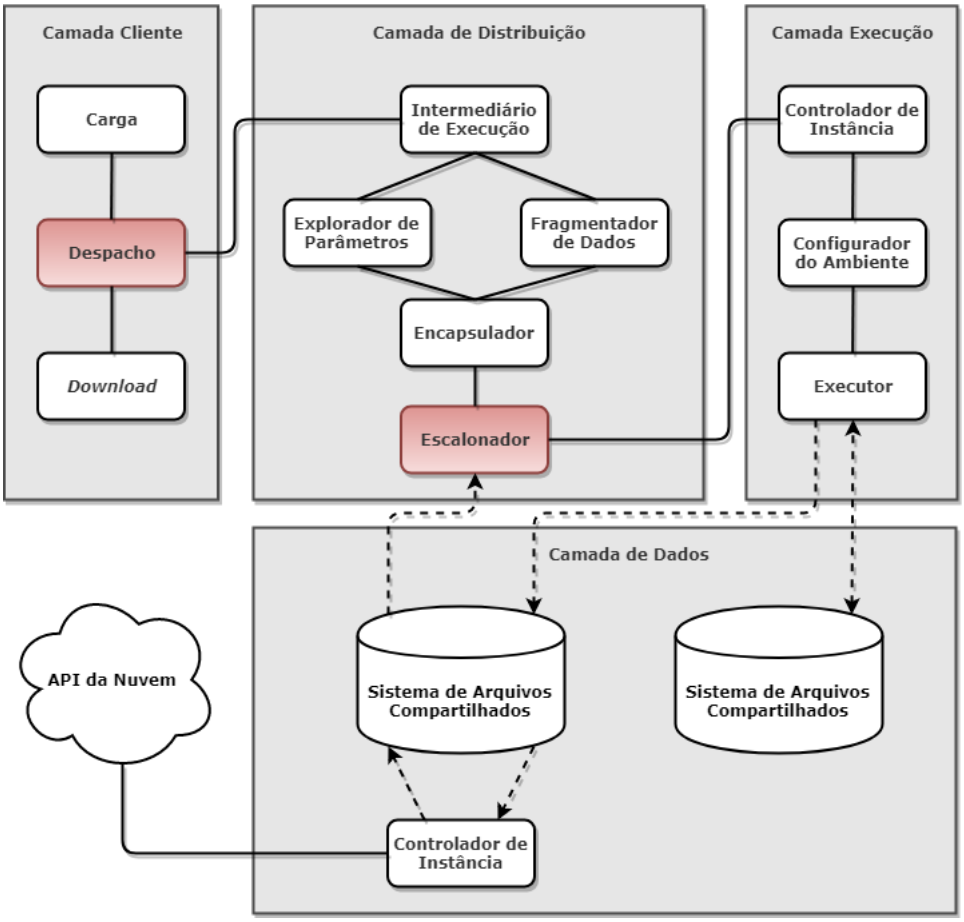

Figura 3.2: *Arquitetura conceitual do SciCumulus [7] adaptado com as extensões destacadas*

A Figura 3.2 apresenta a arquitetura conceitual original do *SciCumulus* através de camadas e componentes explicados no Capítulo 2 na seção 2.5, sendo que os componentes em destaque representam as extensões propostas pelo *SciCumulus4RL* nesta dissertação.

A extensão do componente despacho proposto pelo *SciCumulus4RL* permite que ele

seja capaz de identificar quando deve seguir ou não o plano de escalonamento baseado em reforço definido pelo *WorkflowSim4RL*. Para isso, foi necessário acrescentar uma nova *tag*, chamada de *algorithm*, no arquivo *XML* de entrada do *SciCumulus4RL* que define a especificação e a configuração do experimento. Assim, quando esta *tag* é preenchida com o valor *q-learning*, o *SciCumulus4RL* utiliza o novo escalonador que segue o plano de escalonamento definido pelo *WorkflowSim4RL*. Caso esta *tag* não esteja no arquivo *XML* de entrada do *SciCumulus4RL* ou com outro valor qualquer, o *SciCumulus4RL* utiliza o seu escalonador padrão.

Com a extensão do componente escalonador oferecido pelo *SciCumulus4RL* foi possível criar um novo escalonador capaz de seguir o plano de escalonamento gerado pelo *WorkflowSim4RL*. Também foi criada uma nova tabela no banco de dados do *SciCumulus4RL* que armazena o plano de escalonamento de um experimento específico, juntamente com o seu identificador. Dessa forma, o novo escalonador do *SciCumulus4RL* reconhece qual experimento está sendo executado no momento e encontra nesta tabela o plano de escalonamento correspondente. Entretanto, o carregamento desse plano de escalonamento não é feito diretamente pelo *SciCumulus4RL*. Ela ocorre de forma automatizada por aplicação auxiliar desenvolvida exclusivamente para isso nesta dissertação. Esta aplicação auxiliar tem a capacidade de ler o arquivo gerado pelo *WorkflowSim4RL* contendo o plano de escalonamento e inserir ou atualizar a tabela o banco de dados do *SciCumulus4RL* com essas informações.

Como informado no Capítulo 2 na seção 2.5, na execução do *SciCumlus* não existe uma centralização da execução do *workflow* com distribuição das ativações dentre as máquinas virtuais. Cada máquina virtual do experimento, ou seja, cada $vm \in$ ao conjunto VM, executa o *SciCumulus*. Quando uma máquina fica ociosa, ela solicita uma ativação disponível para executá-la. Dessa forma, o escalonador é executado em todas as máquinas virtuais definidas no arquivo de configuração. O novo escalonador proposto pelo *SciCumulus4RL* identifica em qual máquina virtual ele está sendo executado e percorre a lista de ativações disponíveis para execução. Para cada ativação faz-se uma consulta na tabela do banco de dados contendo o plano de escalonamento informando o identificador do experimento, ativação e a sua máquina virtual. Caso encontre a tupla na tabela do banco de dados que corresponde a essa consulta, a ativação é executada pela máquina virtual. Caso não encontre, a máquina permanece ociosa aguardando a liberação de uma ativação que possa executar. Este processo é repetido até que ocorra o fim da execução do *workflow*. Com isso, o *SciCumulus4RL* garante que as máquinas virtuais só executem as ativações que estão definidas no plano de escalonamento para que elas as executem,

impossibilitando, por exemplo, de uma máquina virtual executar uma ativação que esteja definido para outra máquina virtual no plano de escalonamento.

Após criar diversas episódios simulados no *WorkflowSim4RL* é possível executar este mesmo plano de escalonamento no *SciCumulus4RL* se beneficiando de ter um plano de escalonamento sugerido pelo aprendizado por reforço que tem como objetivo diminuir o tempo de execução, assim como, o custo financeiro envolvido na execução do experimento. Afinal, os serviços em ambiente de nuvens são cobrados baseados nos recursos consumidos *(pay-per-use)*. Dessa forma, quanto maior for a eficiência do escalonamento, menor é o desperdício de tempo e financeiro.

Capítulo 4

Avaliação Experimental

Este capítulo apresenta as configurações utilizadas para a execução de um *workflow* científico no *WorkflowSim4RL* e no *SciCumulus4RL*, bem como os resultados obtidos. Assim, a ideia central deste capítulo é avaliar a eficiência do plano de escalonamento gerado pelo *WorkflowSim4RL* através do *SciCumulus4RL*. Portanto, avalia-se nesta dissertação os resultados obtidos pelo *WorkflowSim4RL* no *SciCumulus4RL*.

Para a avaliação foram executados um conjunto de experimentos simulados no *WorkflowSim4RL* e alguns destes experimentos foram escolhidos para serem executados pelo *SciCumulus4RL* seguindo o plano de escalonamento definido pelo *WorkflowSim4RL*. Para isso, foi adotado o *workflow* científico ***Montage*** por ser um *workflow* amplamente conhecido e usualmente utilizado em trabalhos relacionados a este tema. O *Montage* é *workflow* de astronomia criado pela NASA que a partir de várias fotos do espaço é capaz de gerar uma única foto representando um mosaico composto pelas fotos iniciais. A versão escolhida foi a m101, isto é, com a região do espaço m101 e com 50 ativações.

Como o foco dessa dissertação é voltado para ambientes de nuvens, ou seja, ambientes heterogêneos, foram escolhidos dois tipos de máquinas virtuais disponíveis na Amazon AWS, a **t2.micro** com 1 vCPU (CPU virtualizada) e com 1 GB de memória RAM e a **t2.2xLarge** com 8 vCPUs e com 16 GB de memória RAM para a análise de execução do experimento. Assim, o SciCumulus criou instâncias destes dois tipos de máquina em suas execuções, enquanto o WorkflowSim4RL foi executado em uma máquina com processador Intel Core i5-7200U @ 2.50GHz 2.71GHz, com 8 GB de memória RAM e com Sistema Operacional Linux Mint, simulando as configurações descritas na Tabela 4.1.

Configuração das máquinas virtuais da Amazon EC2						
Id da Configuração	VMs VMs	VM Micro VMs Micro	VM Large 2xLarge	Ids das Vms Micro	Ids das Vms 2xLarge	Qtd de vCPUs
1	9	8	1	$\{0, \ldots, 7\}$	8	16 núcleos
2	11	8	3	$\{0, \ldots, 7\}$	$\{8, \ldots, 10\}$	32 núcleos
3	15	8	7	$\{0, \ldots, 7\}$	$\{8, \ldots, 14\}$	64 núcleos

Tabela 4.1: Configuração das máquinas da Amazon EC2 simuladas no *WorkflowSim4RL*

Para obter um melhor plano de escalonamento gerado pelo *WorkflowSim4RL*, cada experimento executou 100 episódios com μ igual 0.5 e variou conforme o conjunto de máquinas definido na Tabela 4.1. Além disso, foram variados os 3 parâmetros do *Q-Learning*, α que corresponde a taxa de aprendizado, γ que é o peso atribuído à recompensa futura e ϵ que representa a probabilidade de decidir entre explotação ou exploração. Cada parâmetro assumiu um valor no conjunto $\{0.1, 0.5, 1.0\}$ a cada experimento. Com isso, foram formados 27 variações distintas para uma mesma configuração de máquinas e como as máquinas variaram de acordo com a Tabela 4.1, ao todo foram obtidos 81 resultados. Para cada resultado foi obtido o tempo de aprendizado descrito na Tabela 4.2 e o tempo de simulação representados na Tabela 4.3.

```
for a in 0.1 0.5 1.0
do
    for g in 0.1 0.5 1.0
    do
        for e in 0.1 0.5 1.0
        do
            for i in $(seq 1)
            do
                for j in $(seq 3)
                do
                    for k in $(seq 100)
                    do
                    echo "Run machine config $j - episode $k... with alpha $a, gamma $g, epsilon $e"
                    java -cp WorkflowSim.jar
                    org.uff.qlearning.WorkflowSimReinforcementLearningScheduleExample $i $j $a $g $e
                    done
                done
            done
        done
    done
done
```

Figura 4.1: *Arquivo bash para execução dos experimentos no WorkflowSim4RL*

Devido ao fato do experimento ter que ser executado diversas vezes, surgiu a necessidade de criar um mecanismo que automatizasse o processo. Dessa forma, foi criado um arquivo *bash* conforme representado pela Figura 4.1 que permite executar sequencialmente

todos os experimentos no *WorkflowSim4RL*. Para isso, considere a como o α do algoritmo do *RFLScheduling*, g representando o γ do algoritmo do *RFLScheduling*, e representando o ϵ do algoritmo do *RFLScheduling*, i como o *workflow* científico *Montage*, j representa o id da configuração baseado na Tabela 4.1 e k a quantidade de episódios utilizado no experimento.

Parâmetros			Tempo de aprendizado		
alpha	gamma	epsilon	16 vCPUs	32 vCPUs	64 vCPUs
0.1	0.1	0.1	93.0	98.0	119.0
0.1	0.1	0.5	90.0	99.0	114.0
0.1	0.1	1.0	88.0	100.0	116.0
0.1	0.5	0.1	92.0	100.0	110.0
0.1	0.5	0.5	88.0	96.0	117.0
0.1	0.5	1.0	96.0	96.0	108.0
0.1	1.0	0.1	83.0	96.0	109.0
0.1	1.0	0.5	88.0	96.0	112.0
0.1	1.0	1.0	86.0	97.0	110.0
0.5	0.1	0.1	91.0	100.0	116.0
0.5	0.1	0.5	91.0	94.0	118.0
0.5	0.1	1.0	87.0	95.0	119.0
0.5	0.5	0.1	93.0	97.0	115.0
0.5	0.5	0.5	93.0	95.0	120.0
0.5	0.5	1.0	89.0	97.0	120.0
0.5	1.0	0.1	78.0	93.0	108.0
0.5	1.0	0.5	99.0	93.0	107.0
0.5	1.0	1.0	82.0	86.0	106.0
1.0	0.1	0.1	91.0	93.0	108.0
1.0	0.1	0.5	90.0	93.0	104.0
1.0	0.1	1.0	87.0	92.0	108.0
1.0	0.5	0.1	89.0	92.0	105.0
1.0	0.5	0.5	80.0	89.0	107.0
1.0	0.5	1.0	89.0	96.0	107.0
1.0	1.0	0.1	79.0	90.0	107.0
1.0	1.0	0.5	92.0	89.0	107.0
1.0	1.0	1.0	86.0	90.0	110.0

Tabela 4.2: Tempo de aprendizado dos experimentos em segundos no *WorkflowSim4RL*

Parâmetros			Tempos simulados		
alpha	gamma	epsilon	16 vCPUs	32 vCPUs	64 vCPUs
0.1	0.1	0.1	663.94153	595.16841	671.06544
0.1	0.1	0.5	848.71421	594.01914	796.02482
0.1	0.1	1.0	752.95727	839.18546	592.58062
0.1	0.5	0.1	749.42662	670.53755	665.96669
0.1	0.5	0.5	905.13440	891.55308	746.14164
0.1	0.5	1.0	852.55245	695.53101	537.25396
0.1	1.0	0.1	259.02589	334.16751	256.52062
0.1	1.0	0.5	542.93981	416.73384	440.44978
0.1	1.0	1.0	829.70283	904.39862	585.58822
0.5	0.1	0.1	822.41175	670.84354	814.45827
0.5	0.1	0.5	923.86923	695.94739	672.85318
0.5	0.1	1.0	924.44982	737.26139	770.37092
0.5	0.5	0.1	845.64167	672.09372	641.31525
0.5	0.5	0.5	824.88087	747.06360	594.10582
0.5	0.5	1.0	928.42364	751.88346	774.29824
0.5	1.0	0.1	486.68072	258.06918	332.70493
0.5	1.0	0.5	697.33701	643.53614	438.81466
0.5	1.0	1.0	830.05115	827.60478	510.21845
1.0	0.1	0.1	642.84260	801.82828	486.07712
1.0	0.1	0.5	898.03071	723.51203	497.66092
1.0	0.1	1.0	775.77633	836.65385	693.54991
1.0	0.5	0.1	817.88112	821.89452	639.98802
1.0	0.5	0.5	847.97787	846.69140	412.29261
1.0	0.5	1.0	905.85435	698.15094	652.81855
1.0	1.0	0.1	259.31386	333.56856	279.72661
1.0	1.0	0.5	663.30534	641.21227	432.01269
1.0	1.0	1.0	752.69346	851.41301	511.38465

Tabela 4.3: Tempo obtido pela simulação dos experimentos em segundos no *WorkflowSim4RL*

Após a execução dos experimentos no *WorkflowSim4RL* foram escolhidos os 3 melhores planos de escalonamento para cada configuração de máquina baseado nos tempos de simulação. A tabela 4.4 apresenta os planos escolhidos para configuração de 16 vCPU.

A tabela 4.5 apresenta os planos escolhidos para configuração de 32 vCPU. E por fim, a tabela 4.6 apresenta os planos escolhidos para configuração de 64 vCPU. Esses planos de escalonamento foram executados pelo *SciCumulus4RL* e para avaliar seus desempenhos foram comparados com o algoritmo FCFS, conforme é demostrado na Tabela 4.4, na Tabela 4.5 e na Tabela 4.6.

Tempo de execução do *Montage* em 16 vCPU					
Algoritmo	vCPUs	Alpha	Gamma	Epsilon	Tempo Total
FCFS	16				00:03:09.625
Q-Learning	16	1.0	1.0	0.1	00:03:17.483
Q-Learning	16	0.1	1.0	0.1	00:03:29.584
Q-Learning	16	0.5	1.0	0.1	00:03:37.552

Tabela 4.4: Tempo de execução do *Montage* no *SciCumulus4RL* para 16 vCPUs

Tempo de execução do *Montage* em 32 vCPU					
Algoritmo	vCPUs	Alpha	Gamma	Epsilon	Tempo Total
Q-Learning	32	0.5	1.0	0.1	00:03:17.395
Q-Learning	32	0.1	1.0	0.1	00:03:41.148
FCFS	32				00:03:48.892
Q-Learning	32	1.0	1.0	0.1	00:03:53.511

Tabela 4.5: Tempo de execução do *Montage* no *SciCumulus4RL* para 32 vCPUs

Tempo de execução do *Montage* em 64 vCPU					
Algoritmo	vCPUs	Alpha	Gamma	Epsilon	Tempo Total
Q-Learning	64	0.5	1.0	0.1	00:03:23.26
Q-Learning	64	0.1	1.0	0.1	00:03:35.23
Q-Learning	64	1.0	1.0	0.1	00:03:41.918
FCFS	64				00:03:42.675

Tabela 4.6: Tempo de execução do *Montage* no *SciCumulus4RL* para 64 vCPUs

Comparando os planos de escalonamento gerados pelo escalonador do *Workflow-Sims4RL* com o plano de escalonamento obtido pelo escalonador *FCFS* para cada uma das 3 configurações de experimento é possível encontrar algumas diferenças de como os escalonadores atuam. Para melhor compreender essa comparação, é preciso lembrar que conforme descrito na tabela 4.1 os identificadores entre 0 e 7 correspondem as máquinas

virtuais do tipo micro e os identificadores maiores que 7 correspondem as máquinas virtuais do tipo 2xLarge. No plano de escalonamento definido pelo escalonador *FCFS* para a configuração 1, ou seja, com 16 vCPUs (vide tabela 4.7), as 9 ativações iniciais são distribuídas sequencialmente entre as 8 máquinas virtuais disponíveis. Nas demais configurações (consulte as tabelas 4.8 e 4.9) este padrão também aparece. Então, é possível perceber que o escalonador *FCFS* não faz distinção alguma do quanto essas ativações podem ser computacionalmente exaustivas e se necessitam serem alocadas em máquinas virtuais mais robustas. Analisando esse comportamento conclui-se que possivelmente, deve haver ativações prontas para serem executadas e máquinas virtuais ociosas. Com isso, o escalonador *FCFS* faz a alocação dessas ativações nessas máquinas virtuais ociosas. O mesmo já não ocorre nos planos de escalonamentos obtidos pelo *WorkflowSims4RL* que tendem a efetuar essas alocações nas máquinas virtuais mais robustas, pois para a configuração 1, conforme é apresentada na tabela 4.7, é possível observar o predomínio de alocação dessas ativações na máquina virtual do tipo 2xLarge. Outro pronto relevante detectado através da análise dos planos de escalonamento é que as ativações que devem exigir maior recurso computacional são aproximadamente as 18 primeiras ativações, porque é nessa região onde concentra o maior número de alocação para a máquina virtual do tipo 2xLarge, e este comportamento se repete em todas as configurações de acordo com as tabelas 4.8 e 4.9. Portanto, através da análise dos planos de escalonamento fornecidos pelo escalonador do *WorkflowSims4RL* em comparação com o plano de escalonamento definido pelo escalonador *FCFS*, nota-se que ocorreu uma predileção em determinadas alocações de ativações em máquinas virtuais. Estas decisões de alocação são reflexos do aprendizado adquirido através do reforço (recompensa/punição).

Planos de escalonamento para o *Montage* com 16 vCPU										
Id da ativação	Id da máquina virtual									
	FCFS	α 1.0	γ 1.0	ϵ 0.1	α 0.5	γ 1.0	ϵ 0.1	α 1.0	γ 1.0	ϵ 0.1
0	0		4			8			4	
1	0		8			8			8	
2	1		8			4			8	
3	2		8			1			8	
4	3		8			8			8	
5	4		8			3			8	
6	5		8			8			8	
7	6		8			8			8	
8	7		8			8			8	
9	8		8			8			8	
10	8		8			8			8	
11	0		8			8			8	
12	1		8			8			8	
13	0		7			7			7	
14	1		8			8			8	
15	1		8			8			8	
16	2		8			8			8	
17	3		7			7			7	
18	4		8			8			8	
19	5		6			6			6	
20	6		7			7			7	
21	7		5			5			5	
22	8		8			8			8	
23	8		4			4			4	
24	3		5			5			5	
25	4		3			3			3	
26	2		6			6			6	
27	7		2			2			2	
28	6		7			7			7	
29	5		1			1			1	
30	1		8			8			8	
31	8		0			0			0	
32	1		4			4			4	
33	4		2			2			2	
34	2		3			3			3	
35	3		7			7			7	
36	6		8			8			8	
37	8		1			1			1	
38	8		8			8			8	
39	7		6			6			6	
40	2		5			5			5	
41	3		4			4			4	
42	4		3			3			3	
43	7		0			0			0	
44	5		5			5			5	
45	6		6			6			6	
46	8		2			2			2	
47	1		1			1			1	
48	5		7			7			7	
49	1		8			8			8	

Tabela 4.7: Planos de escalonamento do *Montage* gerados pelo *WorkflowSim4RL* para 16 vCPUs

Planos de escalonamento para o *Montage* com 32 vCPU										
Id da ativação	Id da máquina virtual									
	FCFS	α 0.1	γ 1.0	ϵ 0.1	α 0.5	γ 1.0	ϵ 0.1	α 1.0	γ 1.0	ϵ 0.1
0	0		7			10			9	
1	0		10			10			7	
2	1		10			10			10	
3	2		10			10			10	
4	3		10			10			10	
5	4		10			10			10	
6	5		10			10			10	
7	6		6			10			10	
8	7		10			10			10	
9	8		10			10			10	
10	9		10			10			10	
11	0		10			10			10	
12	1		10			10			10	
13	0		9			9			9	
14	1		5			10			6	
15	1		10			10			10	
16	2		10			10			10	
17	3		9			9			9	
18	4		9			9			9	
19	5		8			6			8	
20	6		10			10			10	
21	7		7			8			7	
22	8		9			9			9	
23	9		6			7			6	
24	10		10			8			10	
25	8		5			5			5	
26	3		8			10			8	
27	10		4			4			4	
28	9		7			6			2	
29	4		3			3			7	
30	2		9			9			9	
31	7		2			2			3	
32	6		10			8			10	
33	8		8			10			8	
34	5		1			1			1	
35	9		0			0			0	
36	3		5			4			5	
37	4		4			5			4	
38	2		10			3			10	
39	6		7			6			2	
40	10		3			9			7	
41	8		2			10			3	
42	9		9			8			9	
43	1		8			2			8	
44	7		6			7			6	
45	10		1			1			1	
46	5		0			4			0	
47	8		4			0			4	
48	10		7			5			2	
49	1		10			10			10	

Tabela 4.8: Planos de escalonamento do *Montage* gerados pelo *WorkflowSim4RL* para 32 vCPUs

Planos de escalonamento para o *Montage* com 64 vCPU										
Id da ativação	Id da máquina virtual									
	FCFS	α 0.1	γ 1.0	ϵ 0.1	α 0.5	γ 1.0	ϵ 0.1	α 1.0	γ 1.0	ϵ 0.1
0	0	14			14			14		
1	0	14			14			14		
2	1	14			14			14		
3	2	14			14			14		
4	3	14			14			14		
5	4	14			14			14		
6	5	14			2			14		
7	6	14			14			14		
8	7	12			14			14		
9	8	14			14			14		
10	9	14			14			4		
11	0	14			14			14		
12	1	14			14			14		
13	0	13			13			13		
14	1	14			14			14		
15	1	14			14			14		
16	2	14			14			14		
17	3	13			13			13		
18	4	13			13			13		
19	5	12			12			12		
20	6	0			14			14		
21	7	14			11			11		
22	8	13			13			13		
23	9	11			10			10		
24	10	14			5			11		
25	11	10			11			9		
26	12	12			14			14		
27	13	9			9			8		
28	14	8			12			12		
29	14	7			8			7		
30	13	13			13			13		
31	12	6			7			6		
32	1	11			10			10		
33	5	10			11			9		
34	9	5			6			5		
35	6	4			4			4		
36	8	14			14			11		
37	14	9			12			3		
38	3	0			5			1		
39	11	3			9			12		
40	4	8			0			7		
41	2	2			3			2		
42	10	13			1			12		
43	7	12			8			8		
44	11	1			2			14		
45	8	11			9			13		
46	10	9			13			0		
47	13	10			10			10		
48	12	14			14			14		
49	1	14			14			14		

Tabela 4.9: Planos de escalonamento do *Montage* gerados pelo *WorkflowSim4RL* para 64 vCPUs

Capítulo 5

Trabalhos Relacionados

Este capítulo apresenta trabalhos relacionados com esta dissertação, ou seja, abordagens semelhantes para criação de um escalonador de *workflows* científicos baseado em aprendizado por reforço para ambiente de nuvens de computadores. Foram analisados diversos artigos e trabalhos sobre o tema e os trabalhos descritos aqui foram os que, de alguma forma, serviram como ponto de partida ou incentivo para a produção desta pesquisa.

Baseado nestes estudos, este capítulo expõe através de seções um escalonador baseado em modelo de custo multiobjetivo para ambiente de nuvens, um escalonador multiobjetivo para ambiente de grades computacionais, um escalonador baseado aprendizado por reforço para ambiente de grades, um escalonador baseado em algoritmo genético para ambiente de grades computacionais, um escalonador com abordagem multifacetada: algoritmo genético + MPD para ambiente de nuvens e escalonador baseado em aprendizado por reforço usando sistemas de filas.

5.1 Escalonador baseado em modelo de custo multiobjetivo para ambiente de nuvens

O trabalho proposto por Oliveira [7] serviu como alicerce para o desenvolvimento de um escalonador baseado em aprendizado por reforço para execução de *workflows* científicos em ambiente de nuvens usando o Sistemas de Gerência de Workflows Científicos (SGWfC). Desta forma, esta dissertação é uma continuação do trabalho citado, pois o SGWfC, chamado *SciCumulus* apresentado por Oliveira [7] foi estendido nessa dissertação.

Diferentemente desta dissertação, o seu o escalonador é definido através da forma-

lização de um modelo de custo multiobjetivo [7]. Este modelo consiste em uma série de fórmulas que formaliza diversas caraterísticas do ambiente de nuvens. Propriedades de uma máquina virtual, tais como: quantidade de memória, capacidade de processamento juntamente com os tempos médios de execuções anteriores das ativações e índices de retardo computacional da máquina virtual necessitam ser formalizadas. O tempo de execução previsto de uma ativação em uma determinada máquina virtual baseado em histórico de informação também são formalizados neste modelo. A formalização da confiabilidade de cada máquina virtual é dada por uma função probabilística. Além disso, outras características como largura de banda para transferência de dados e o custo monetário também são formalizados. Este modelo permite maximizar mais de um objetivo simultaneamente como tempo de execução e custo financeiro, por exemplo (já que o custo de utilização de máquinas deste ambiente pode ser variável).

Como não existe escalonador baseado em aprendizado por reforço exposto por ele, esta dissertação diverge desse trabalho por propor um escalonador baseado em aprendizado por reforço para ambiente de nuvens onde o aprendizado ocorre em ambiente simulado.

5.2 Escalonador multi-objetivo para ambiente de grades computacionais

O trabalho apresentado por Yu [30] propõe um escalonador de *workflow* científico multiobjetivo para ambiente de grades computacionais, e, assim como no modelo de escalonamento apresentado por Oliveira, [7], esse escalonador é definido por uma série de fórmulas que formaliza diversas caraterísticas do ambiente, sendo que neste trabalho o ambiente modelado é o de grades.

Apesar do foco do trabalho proposto por Yu [30] ser *workflows* científicos, a ausência de um escalonador baseado em aprendizado por reforço torna esse trabalho divergente desta dissertação.

5.3 Escalonador baseado aprendizado por reforço para ambiente de grades

Outro trabalho que teve bastante importância para esta dissertação foi apresentado por Costa *et al.* [6] que visa criar equações de desempenho de máquinas virtuais aplicando aprendizado por reforço em ambiente de grades computacionais. Dessas forma,

as equações de desempenho apresentadas nesta dissertação, bem como, as equações de recompensa/punição foram inspiradas por Costa *et al.* [6]. Todas essas equações são explicadas detalhadamente no Capítulo 2 na seção 3.1.2.

O algoritmo apresentado por Costa *et al.* [6] utiliza-se dos conceitos de aprendizado por reforço e das equações de desempenho e recompensa/punição para propor um escalonamento de tarefas para ambiente de grades computacionais. Sua abordagem é composta por duas fases. Na primeira, pequenas tarefas são alocadas para recursos de grade para serem executadas e com isso, obtendo-se o desempenho dos recursos. Após a conclusão desta fase, as tarefas restantes são classificadas e ordenadas decrescentemente baseadas nos tempos de execução estimados. Também são organizadas em grupos de acordo com o número de recursos. Cada grupo é associado a um recurso de forma que recursos mais eficientes executem tarefas com maior tempo de execução. Dessa forma, as tarefas com maior tempo de execução estimado são alocadas primeiro em cada grupo seguindo por tarefas com menor tempo de execução estimado até que não haja mais tarefas a serem alocadas.

Embora possuam abordagens semelhantes, o trabalho apresentado por Costa *et al.*[6] é voltado para ambiente de grades, não é específico para *workflows* científicos e o seu algoritmo proposto é desconectado do Processo de Decisão de Markov. Outro ponto de divergência é que processo de aprendizagem não ocorre através de simulações.

5.4 Escalonador baseado em algoritmo genético para ambiente de grades computacionais

Yu e Buyya [29] apresenta escalonador para *workflows* científicos baseado em algoritmo genético que tem como objetivo minimizar o tempo de execução baseado em uma restrição orçamentária do usuário para ambiente de grades computacionais.

Através dos conceitos de algoritmo genético, primeiro o escalonador cria um conjunto inicial de planos de escalonamento, chamado de população inicial. Cada plano é denominado de indivíduo e são escolhidos com base em uma equação de avaliação. O mapeamento entre uma ativação e uma máquina virtual deste plano de escalonamento é definido como o cromossomo para essa abordagem de algoritmo genético. Cada indivíduo, ou seja, cada plano de escalonamento é avaliado e pode sofrer uma mutação aleatória, que consiste em uma mudança de alocação de ativação em uma máquina. Portanto essas alterações tornam o escalonador não determinístico. Além disso, são feitos cruzamentos entre indivíduos a

fim de gerar novos descendentes (novos indivíduos) mais adaptados, neste caso, planos de escalonamento mais eficientes. Este processo é repetido diversas vezes com o intuito de encontrar planos de escalonamento que satisfaçam a restrição orçamentária estabelecida pelo usuário.

Por Yu e Buyya [29] apresentaram uma abordagem de escalonamento baseada em algoritmo genético e seu foco ser grades computacionais torna esse trabalho divergente da abordagem proposta por esta dissertação.

5.5 Escalonador com abordagem multifacetada: algoritmo genético + MPD para ambiente de nuvens

O trabalho exposto por Barrett *et al.* [3] propõe a criação de um escalonador de *workflow* para ambiente de nuvens utilizando uma abordagem multifacetada, através de um algoritmo genético que tenta encontrar planos de escalonamentos ótimos e um Processo de Decisão de Markov (MPD) que efetua a escolha entre estes planos de escalonamento com base no estado observado do ambiente. Diferentemente desta dissertação, esse trabalho não é específico para *workflows* científicos, nem utiliza técnicas de aprendizado por reforço para criação do escalonador.

Esse trabalho também utiliza os mesmos conceitos de algoritmo genético que o trabalho proposto por Yu e Buyya [29], porém as funções de avaliação e o algoritmo de escalonamento são diferentes.

Apesar do trabalho apresentado por Barrett *et al.* [3] utilizar o Processo de Decisão de Markov, ele o utiliza de forma diferente desta dissertação. Pois, o Processo de Decisão de Markov apenas tenta encontrar o melhor plano de escalonamento dentre os planos gerados pelo algoritmo genético, quem é que efetivamente faz as tomadas de decisão entre as alocações das ativações em máquinas virtuais no ambiente de nuvem. Portanto, diverge desta dissertação que utiliza o Processo de Decisão de Markov nas tomadas de decisão de alocação de ativações em máquinas virtuais durante o escalonamento.

5.6 Escalonador baseado em aprendizado por reforço usando sistemas de filas

Peng *et al.* [22] propõe um escalonador de tarefas para ambiente de nuvens computacionais usando um sistema de filas. Para isso, ele explora os seguintes modelos do

sistema de fila para o controle das tarefas: cronograma de tarefas (TSSM), executável de tarefas executadas e transmissão de tarefas (TTSM). Ele otimiza o escalonador de tarefas através do aprendizado por reforço com base nesses modelos e acelera o progresso de aprendizado utilizando a agregação de estados. Entretanto, esses modelos se diferenciam do objeto desta dissertação que é escalonamento para *worklow* científicos. Os SGWfC já são capazes de gerenciar *worklow* científicos sem a necessidade de criação desses modelos. Diferentemente desta dissertação, a avaliação do escalonamento proposto por Peng *et al.* [22] é dada pelo tempo de resposta das filas, ou seja, quanto mais rápida a tarefa sair da fila, executar e informar o fim de sua execução, mais eficiente foi o escalonamento.

Semelhante a este trabalho de dissertação, o escalonador baseado em aprendizado por reforço proposto por Peng *et al.* [22] também utiliza o algoritmo *Q-Learning*. Porém, funções de avaliação de recompensa/punição são definidas de modo diferente. Além de possuir três respostas de recompensas/punição: recompensa quando a avaliação mostrar respostas positiva, punição quando avaliação mostrar resposta negativa, neutra quando a avaliação não mostrar melhora nem piora do desempenho. Como nesse trabalho o aprendizado ocorre ao longo da execução das tarefas sem que haja um aprendizado prévio, ele utiliza-se de tecnologias de agregação de estado para acelerar o progresso da aprendizagem.

Portanto, mesmo que o trabalho de Peng *et al.* [22] utilize no seu escalonador o aprendizado por reforço e o algoritmo *Q-Learning*, ele é desconectado do conceito de *workflow* científicos, bem como os SGWfCs.

Capítulo 6

Conclusão

Este capítulo apresenta as conclusões gerais desta dissertação além das principais contribuições e limitações. Também apresenta as principais perspectivas de trabalhos futuros que podem ser desenvolvidos como consequência direta desta dissertação.

6.1 Contribuições

A abordagem proposta por esta dissertação é inspirada na função de desempenho de máquina virtual aplicando aprendizado por reforço em um ambientes de grade[6], e propõe um escalonador baseado em aprendizado por reforço para execução de *workflows* científico em ambientes de nuvens de computadores, considerando ambientes heterogêneos, sendo que o aprendizado ocorre em ambiente simulado. Afinal, execuções de $workflows$ científicos comumente demandam grande capacidade de processamento e envolve grande volume de dados. Por isso, ambientes de nuvens tem como vantagens as características da elasticidade e da personalização de máquinas virtuais, e dessas forma, é possível adequar a quantidade de máquinas virtuais (VMs) utilizadas para execução de um determinado $workflow$ científico, visto que, em ambientes de nuvem o cientista só paga pelos recursos que consome *(pay-per-use)*. Assim, um SGWfCs eficiente deve ter a capacidade de distribuir as execuções do *workflow* científico nestes ambientes de forma eficiente evitando o desperdício de recursos computacionais e, com isso, evitar o desperdício financeiro e de tempo.

Neste contexto, a abordagem apresentada nesta dissertação, em que um escalonador baseado em aprendizado por reforço que efetua seu aprendizado em ambiente simulado e depois o aplica em um ambiente real, traz uma vantagem ao cientista, pois, além de o escalonador seguir um plano de escalonamento mais eficiente para ambiente com maior

número de processadores virtuais (32 vCPUs e 64 vCPUs), todo aprendizado pode ser feito localmente ou em uma única máquina virtual no ambiente de nuvem. Como consequência, a etapa do processo de aprendizagem torna-se gratuito ou pouco oneroso. Outra vantagem é que o aprendizado por ser uma simulação ocorre em um tempo muito curto, enquanto execuções reais dos *worflows* científicos costumam ser bastante demorados.

Visando prover maior suporte aos cientistas em experimentos científicos em ambientes de alto desempenho, dado que este tema tem sido abordado em diversos congressos e conferências, seja ela nacional ou internacional, esta pesquisa de dissertação apresentou esta abordagem com seus resultados que provê este apoio.

6.2 Limitações

Assim como todo e qualquer trabalho científico, esta dissertação possui limitações. O SciCumulus4RL poderia funcionar de forma mais integrada como o WorkflowSim4RL. Ou seja, se ambos trabalhassem com maior integração e transparência permitindo que o plano de escalonamento gerado pelo workflowSim4RL fosse inserido no repositório do SciCumlus4RL de forma automatizada, sem a necessidade que este processo seja feito por outra aplicação e/ou sem a necessidade do usuário tivesse que efetuar esta atividade para usar o escalonador baseado em apredizado por reforço. Outra limitação é que os resultados obtidos nas execuções reais, isto é pelo SciCumlus4RL, não contribuem para o aprendizado do escalonador no WorkflowSim4RL. Atualmente, caso queira fazer essa retroalimentação de informações é necessário alterar manualmente no XML do WorkflowSim4RL com os novos valores obtidos.

6.3 Trabalhos Futuros

Um escalonador que se baseia em aprendizado por reforço não é uma tarefa fácil, muito menos trivial. Pelo contrário, é uma tarefa complexa e que demanda tempo, principalmente por tratar-se de um problema de escalonamento que é um problema de complexidade NP (não polinomial). Com isso, considerando o estágio atual do desenvolvimento e as necessidades dos cientistas no tema, algumas das perspectivas de trabalhos futuros são apresentadas a seguir:

i) A automatização do processo de migração do plano de escalonamento entre o WorkflowSim4RL e o SciCumulus4RL que poderia ser implementado da seguinte forma:

ao executar um *worflow* científico no SciCumulus4RL, este deveria primeiro executar internamente o WorkflowSim4RL a fim de obter o plano de escalonamento baseado em aprendizado por reforço. Em seguida, deveria executar o *workflow* normalmente utilizando o plano de escalonamento obtido pelo WorkflowSim4RL. Além disso, os valores obtidos na execução real do SciCumulus4RL poderiam contribuir realimentando os parâmetros de entrada do WorkflowSim4RL, aumentando a precisão e confiabilidade dos resultados. Dessa forma, ficaria mais transparente e mais prático para o usuário (cientista) a execução do *workflow* científico usando o escalonador baseado em aprendizado por reforço, pois o processo de execução do *workflow* com o este novo escalonador não seria diferente do processo de execução com os demais escalonadores. Afinal, a sinalização do uso ou não do novo escalonador já ocorre através de uma *tag* no XML de entrada do SCiCumulus4RL, então bastaria mantê-la.

ii) Outra possibilidade de ampliação deste trabalho seria encontrar uma melhor curva de aprendizagem. Para isso, seria necessário reexecutar o *workflow* científico aumentando o número de episódios, variando gradualmente os parâmetros α, γ e ϵ do algoritmo Q-Learning, assim como a função de recompensa/punição. Posteriormente, analisar de forma profunda os resultados buscando padrões que levasse a um escalonamento mais eficiente.

iii) Por fim, também compreende-se que o WorflowSim4RL poderia propor uma abordagem mais flexível relacionado às métricas de eficiência do escalonamento, pois além da métrica utilizada nesta dissertação que é minimizar o tempo de execução, poderia permitir outras métricas como minimizar custo financeiro da execução do *workflow* baseado nas máquinas virtuais disponíveis, minimizar taxas de falhas, entre outras.

Referências

[1] *Oxford Dictionary of English*. Oxford University Press, 2004.

[2] Arabnejad, H., Barbosa, J. G., Prodan, R. Low-time complexity budget-deadline constrained workflow scheduling on heterogeneous resources. *Future Generation Comp. Syst. 55* (2016), 29–40.

[3] Barrett, E., Howley, E., Duggan, J. A learning architecture for scheduling workflow applications in the cloud. Em *2011 IEEE Ninth European Conference on Web Services* (2011), IEEE.

[4] Buneman, P., Khanna, S., Tan, W.-C. Why and Where: A Characterization of Data Provenance. Em *International Conference on Database Theory, 2001* (2001), p. 316–330.

[5] Chen, W., Deelman, E. Workflowsim: A toolkit for simulating scientific workflows in distributed environments. Em *E-science (e-science), 2012 IEEE 8th International Conference on* (2012), IEEE, p. 1–8.

[6] Costa, B. F., Mattoso, M., Dutra, I. Applying reinforcement learning to scheduling strategies in an actual grid environment. *International Journal of High Performance Systems Architecture 2*, 2 (2009), 116–128.

[7] De Oliveira, D., Ogasawara, E., Baião, F., Mattoso, M. Scicumulus: A lightweight cloud middleware to explore many task computing paradigm in scientific workflows. Em *Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on* (2010), IEEE, p. 378–385.

[8] Deelman, E., Gannon, D., Shields, M., Taylor, I. Workflows and e-Science: An overview of workflow system features and capabilities. *Future Generation Computer Systems 25*, 5 (2009), 528–540.

[9] Deelman, E., Mehta, G., Singh, G., Su, M.-H., Vahi, K. Pegasus: mapping large-scale workflows to distributed resources. Em *Workflows for e-Science*. Springer, 2007, p. 376–394.

[10] Freire, J., Koop, D., Santos, E., Silva, C. T. Provenance for Computational Tasks: A Survey. *Computing in Science & Engineering* (Maio/Junho 2008), 20–30.

[11] Lima, K. E. C., Teixeira, F. M. A epistemologia e a história do conceito experimento/experimentação e seu uso em artigos científicos sobre ensino das ciências.

[12] Liu, J., Pacitti, E., Valduriez, P., de Oliveira, D., Mattoso, M. Multi-objective scheduling of Scientific Workflows in multisite clouds. *Future Generation Comp. Syst. 63* (2016), 76–95.

[13] Liu, J., Pacitti, E., Valduriez, P., Mattoso, M. A Survey of Data-Intensive Scientific Workflow Management. *Journal of Grid Computing 13*, 4 (Dec 2015), 457–493.

[14] Liu, J., Pacitti, E., Valduriez, P., Mattoso, M. Scientific Workflow Scheduling with Provenance Data in a Multisite Cloud. *T. Large-Scale Data- and Knowledge-Centered Systems 33* (2017), 80–112.

[15] Liu, J., Sousa, V. S., Pacitti, E., Valduriez, P., Mattoso, M. Scientific Workflow Partitioning in Multisite Cloud. Em *Euro-Par 2014: Parallel Processing Workshops - Euro-Par 2014 International Workshops, Porto, Portugal, August 25-26, 2014, Revised Selected Papers, Part I* (2014), p. 105–116.

[16] Mathá, R., Ristov, S., Prodan, R. A Simplified Model for Simulating the Execution of a Workflow in Cloud. Em *Euro-Par 2017: Parallel Processing - 23rd International Conference on Parallel and Distributed Computing, Santiago de Compostela, Spain, August 28 - September 1, 2017, Proceedings* (2017), p. 319–331.

[17] Mattoso, M., Werner, C., Travassos, G., Braganholo, V., Murta, L., Ogasawara, E., Oliveira, F., Martinho, W. Desafios no apoio à composição de experimentos científicos em larga escala. *Seminário Integrado de Software e Hardware, SEMISH 9* (2009), 36.

[18] Mitchell, T. M. *Machine learning*. McGraw Hill series in computer science. McGraw-Hill, 1997.

[19] Ocaña, K. A., de Oliveira, D., Ogasawara, E., Dávila, A. M., Lima, A. A., Mattoso, M. Sciphy: A cloud-based workflow for phylogenetic analysis of drug targets in protozoan genomes. Em *Advances in Bioinformatics and Computational Biology*. Springer, 2011, p. 66–70.

[20] Ogasawara, E., Dias, J., Oliveira, D., Porto, F., Valduriez, P., Mattoso, M. An algebraic approach for data-centric scientific workflows. *Proc. of VLDB Endowment 4*, 12 (2011), 1328–1339.

[21] Oliveira, D., Ogasawara, E., Ocaña, K., Baião, F., Mattoso, M. An adaptive parallel execution strategy for cloud-based scientific workflows. *Concurrency and Computation: Practice and Experience 24*, 13 (2012), 1531–1550.

[22] Peng, Z., Cui, D., Zuo, J., Li, Q., Xu, B., Lin, W. A learning architecture for scheduling workflow applications in the cloud. Em *National Natural Science Foundation of China* (2015).

[23] Samuel, A. L. Some studies in machine learning using the game of checkers. Em *IBM Journal of Research and Development, 2000* (2000), p. 206–226.

[24] Silva, C. E. P. Captura de dados de proveniência de worflows científicos em nuvens computacionais.

[25] Sutton, R. S., Barto, A. G. *Reinforcement learning: An introduction*, vol. 1. MIT press Cambridge, 1998.

[26] Travassos, G. H., Barros, M. O. Contributions of in virtuo and in silico experiments for the future of empirical studies in software engineering. Em *2nd Workshop on Empirical Software Engineering the Future of Empirical Studies in Software Engineering* (2003), p. 117–130.

[27] Vaquero, L. M., Rodero-Merino, L., Caceres, J., Lindner, M. A Break in the Clouds: Towards a Cloud Definition. *SIGCOMM Comput. Commun. Rev. 39*, 1 (dezembro de 2008), 50–55.

[28] Wang, K., Zhou, X., Li, T., Zhao, D., Lang, M., Raicu, I. Optimizing load balancing and data-locality with data-aware scheduling. Em *2014 IEEE International Conference on Big Data, Big Data 2014, Washington, DC, USA, October 27-30, 2014* (2014), p. 119–128.

[29] Yu, J., Buyya, R. A Budget Constrained Scheduling of Workflow Applications on Utility Grids using Genetic Algorithms. Em *Workshop on Workflows in Support of Large-Scale Science* (2006), IEEE.

[30] Yu, J., Kirley, M., Buyya, R. Multi-objective planning for workflow execution on grids. Em *Proceedings of the 8th IEEE/ACM International conference on Grid Computing* (2007), IEEE Computer Society, p. 10–17.

[31] Zhao, Y., Hategan, M., Clifford, B., Foster, I., Von Laszewski, G., Nefedova, V., Raicu, I., Stef-Praun, T., Wilde, M. Swift: Fast, reliable, loosely coupled parallel computation. Em *Services, 2007 IEEE Congress on* (2007), IEEE, p. 199–206.

[32] Zhao, Y., Raicu, I., Foster, I. Scientific workflow systems for 21st century, new bottle or new wine? Em *Services-Part I, 2008. IEEE Congress on* (2008), IEEE, p. 467–471.