$Workflow Sim4RL$: Aprendizado por Reforço Aplicado ao Escalonamento de *Workflows* Científicos em Nuvens *

Victor Olimpio[1], **André Nascimento**[1], **Aline Paes**[1], **Daniel de Oliveira**[1]

[1]Instituto de Computação – Universidade Federal Fluminense (UFF)
Av. Gal. Milton Tavares de Souza, s/nº
Boa Viagem - Niterói/RJ - Brasil

{victorolimpio, anascimento}@id.uff.br, {alinepaes,danielcmo}@ic.uff.br

Resumo. *Ambientes de Nuvem são amplamente utilizados para a execução de experimentos científicos modelados como workflows. Muitos Sistemas de Gerência de Workflows Científicos (SGWfC) oferecem apoio a execuções na nuvem. Em geral, esses SGWfCs possuem um escalonador de atividades que segue um modelo de custo bem definido. Um problema de se criar modelos de custos para escalonar workflows na nuvem é que precisamos modelar manualmente grande parte das características do ambiente. Entretanto, eventos como migração de máquinas virtuais ou flutuações de desempenho não são facilmente modelados. De forma a gerar um escalonamento sem a necessidade de modelar todas as características do ambiente de antemão, esse artigo propõe um simulador chamado WorkflowSim4RL que utiliza técnicas de Aprendizado por Reforço (AR - da área de aprendizado de máquina) para descobrir como melhor escalonar as ativações do workflow. Por meio do AR o escalonador não necessita conhecer todas as características do ambiente e do workflow sendo executado. Experimentos com traces de workflows reais mostraram a viabilidade da proposta.*

1. Introdução

Nas últimas décadas o modo de se 'fazer ciência' mudou radicalmente. Experimentos científicos passaram a se beneficiar fortemente da infraestrutura computacional existente. Uma nova categoria de experimentos surgiu: os experimentos *in silico* [Travassos and Barros 2003]. Nesse tipo de experimento, todo o processo de experimentação é baseado em simulações computacionais. Muitas dessas simulações são compostas pela execução de diversas aplicações científicas que possuem dependência de dados entre si, *i.e.*, os dados produzidos por uma aplicação são consumidos pela próxima aplicação no fluxo [Mattoso et al. 2009].

Apesar de existirem diversas formas de implementar o encadeamento de programas dentro desses experimentos (*scripts* Python, *bash scripts*, etc.), uma das formas mais utilizadas são os chamados *workflows* científicos [Deelman et al. 2009]. Os *workflows* científicos permitem aos cientistas modelar as etapas do experimento como um grafo, no qual vértices representam atividades de processamento de dados e arestas representam dependências entre as mesmas. Cada atividade do *workflow* está associada a um programa de computador (ou um serviço *Web*), e cada execução de uma atividade com um determinado conjunto de parâmetros é chamada de *ativação*. Cada ativação processa um conjunto de dados de entrada e produz um conjunto de dados de saída [Ogasawara et al. 2011].

Os *workflows* são gerenciados por sistemas de gerência de *workflows* (SGWfC) que são sistemas complexos que apoiam a composição, execução e análise dos resultados dos

*Este artigo foi financiado parcialmente pelo CNPq, CAPES e FAPERJ

experimentos [Deelman et al. 2009]. Considerando que muitos *workflows* consomem e produzem um grande volume de dados e podem processar durante muito tempo, ambientes de processamento de alto desempenho (PAD) devem ser empregados na execução dos *workflows*. Tradicionalmente, esses experimentos têm sido executados em *clusters* de computadores e grades computacionais, mas nos últimos anos as nuvens de computadores têm ganhado importância no cenário científico.

As nuvens de computadores são capazes de fornecer um ambiente favorável para a execução de *workflows*, pois além da disponibilidade, os cientistas podem explorar a elasticidade, *i.e.* a capacidade de aumentar e/ou diminuir a quantidade de máquinas virtuais (VMs) criadas para executar o *workflow*. Na nuvem, o cientista paga apenas pelos recursos que consumir *(pay-per-use)* [Vaquero et al. 2008]. Para que a nuvem possa ser utilizada em sua capacidade máxima, os SGWfCs existentes devem ser capazes de distribuir as execuções dos *workflows* nestes ambientes. Existem diversos SGWfCs que apoiam a execução em nuvens como o Pegasus [Deelman et al. 2007], o Swift/T [Zhao et al. 2007] e o SciCumulus [Oliveira et al. 2012], que foi o SGWfC utilizado neste artigo.

Todos esses SGWfCs possuem escalonadores específicos para o ambiente de nuvem, com modelos de custo definidos de antemão, e considerando critérios como desempenho, custo financeiro, disponibilidade, etc. Cada escalonador é baseado em um modelo de custo que busca formalizar as características do ambiente de nuvem e distribuir as ativações nesse ambiente. Entretanto, existem características do ambiente que são difíceis de serem modeladas ou formalizadas, como por exemplo flutuações de desempenho das VMs e migrações de VMs durante a execução do experimento (*live migration*). Além disso, tais escalonadores dependem de uma boa estimativa do desempenho das ativações (estimativa de tempo de execução), o que pode ser complicado caso o cientista não possua um histórico de execuções anteriores do mesmo *workflow*, ou caso o ambiente não possua recursos de monitoramento. Dessa forma, idealmente, o escalonador dos SGWfCs deveria ser adaptativo ao ambiente, sem que seja imperativo modelar ou formalizar de antemão todas as características do ambiente.

Tal necessidade vai ao encontro a uma área de Aprendizado de Máquina chamada Aprendizado por Reforço [Sutton and Barto 1998]. No Aprendizado por Reforço, um agente específico (no contexto desse artigo o escalonador) deve ser capaz de aprender como se comportar em um ambiente dinâmico (ambiente de nuvem, nesse caso) por meio de interações do tipo "tentativa e erro" (possíveis escalonamentos). A cada tentativa, o agente deve avaliar se a mesma foi uma boa escolha ou não. Em caso de resposta positiva, o agente recebe uma *recompensa*, caso contrário uma *punição*. Tal técnica se encaixa na necessidade anteriormente levantada: o escalonador não precisa conhecer o ambiente e nem estimar o desempenho de ativações, e sim tem que ser capaz de avaliar se uma determinada tentativa (escalonamento) foi boa ou ruim. Tal avaliação é possível uma vez que é simples calcular se uma execução foi mais rápida que a anterior, apresentou um custo financeiro menor, *etc*.

Nesse artigo, apresentamos uma abordagem que propõe o uso de um algoritmo clássico de Aprendizado por Reforço, mais especificamente o *Q-Learning* [Sutton and Barto 1998], aplicada ao problema de escalonamento de *workflows* em recursos computacionais no ambiente de nuvem. Tal abordagem foi implementada no simulador de *workflows* em nuvem chamado $WorkflowSim$ [Chen and Deelman 2012] gerando o $Workflowsim4RL$. A ideia é se aproveitar de um simulador que seja capaz de gerar um plano de escalonamento *a priori* que possa ser informado e seguido pelo SGWfCs. O desempenho da abordagem proposta foi avaliado tanto de forma simulada quanto com a execução do plano de escalonamento gerado

pelo $Workflowsim4RL$ em um ambiente de nuvem real utilizando o SGWfC SciCumulus. Foram realizados avaliações com traces sintéticos de *workflows* e com um *workflow* real da área de astronomia. Os resultados demonstram a viabilidade da abordagem proposta.

O restante deste artigo está estruturado da seguinte maneira. A Seção 2 apresenta referencial teórico. A Seção 3 apresenta a abordagem proposta. A Seção 4 apresenta a avaliação experimental. A Seção 5 aborda trabalhos relacionados, e, finalmente, a Seção 6 conclui o artigo e discute possíveis trabalhos futuros.

2. Referencial Teórico

Nessa seção abordaremos os conceitos básicos de Aprendizado por Reforço, o algoritmo *Q-Learning* utilizado no artigo, e definições básicas referentes a *workflows* científicos.

2.1. Aprendizado por Reforço

Quando estamos aprendendo alguma atividade nova, comumente tendemos a repetir e recombinar os passos para executar determinada tarefa até que consigamos de fato nos convencer de que a atividade foi aprendida. Esse processo acontece dentro de um ciclo de laços de repetições. Além disso, quem está aprendendo, não deve ter influência externa quanto ao que deve ou não fazer, este deve por conta própria descobrir quais decisões tomar para que consiga aumentar as recompensas que recebe do ambiente. E por fim, as ações podem não apenas influenciar a recompensa imediata, mas também situações que se sucederão e, por consequência, suas recompensas. Essas três características - ser por essência fechado em repetições, não ter nenhum tipo de instrução prévia relacionada à escolha de uma ação, e as consequências de ações serem capazes de ter efeito longínquo em situações vindouras - são as três características mais importantes que distinguem o Aprendizado por Reforço [Sutton and Barto 1998] das demais técnicas de Aprendizado de Máquina.

Assim, enquanto as técnicas baseadas em *aprendizado supervisionado* são as mais utilizadas para resolver problemas que requerem Aprendizado de Máquina, elas requerem um conjunto de exemplos anotados previamente, não sendo adequadas para induzir hipóteses em ambientes dinâmicos sem supervisão. Por outro lado, técnicas de *aprendizado não-supervisionado* focam em descobrir padrões ocultos em dados não anotados, mas possuem dificuldade em descobrir funções regendo a dinamicidade do ambiente, incluindo sequências de percepções e ações, e maximizar a valoração de tais ações para alcançar um objetivo. Um dos desafios que surgem no Aprendizado por Reforço, e não nos outros paradigmas, é a troca entre exploração (*exploration*) e explotação (*exploitation*). Para que o agente de Aprendizado por Reforço não perca valores já testados de recompensa, este pode preferir ações que já foram previamente testadas e que se provaram efetivas. No entanto, o agente deve tentar ações que não foram testadas anteriormente a fim de não ficar preso em valores sub-ótimos.

Desta forma, o agente de Aprendizado por Reforço deve explotar o que já conhece a fim de obter recompensas, ao passo que também deve explorar ações que não foram escolhidas anteriormente para que melhores escolhas de ações sejam realizadas futuramente. No entanto, o agente de Aprendizado por Reforço não está isento de falhar em suas escolhas durante os processos de explotação e exploração. Portanto, o agente deve tentar uma variedade de ações e favorecer progressivamente aquelas que oferecem melhores recompensas. Uma estratégia simples pode ser escolher a melhor ação, digamos, 80% das vezes e escolher uma ação nova, selecionada aleatoriamente no restante. Outra característica importante em aprendizado por reforço consiste no agente de aprendizado estar focado diretamente em alcançar um objetivo, interagindo com um ambiente dinâmico, como pode ser observado na Figura 3. Todos os

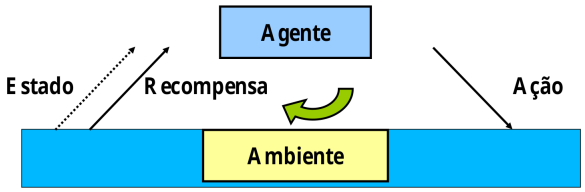

Figura 1. ***Interação do agente de aprendizado por reforço com o ambiente***

agentes de Aprendizado por Reforço possuem objetivos explícitos, são capazes de captar aspectos do ambiente onde se encontram, e podem escolher ações para influenciar seus ambientes, inclusive sendo capazes de atuar em ambientes que envolvem incerteza.

Política Em sistemas de aprendizado envolvendo agentes, dizemos que o agente é o responsável pelo que ele aprende e pelas decisões tomadas, de forma autônoma, enquanto o *ambiente* comprime tudo o que se encontra fora do agente. Ambos interagem entre si, com o agente escolhendo ações a serem executadas no ambiente e este por sua vez respondendo a estas ações. Além disso, o ambiente é capaz de retornar uma recompensa (um valor numérico) para cada ação do agente, que por sua vez tentará maximizar a recompensa recebida ao longo do tempo. A interação entre o agente e o ambiente ocorre ao longo do tempo, por meio de sequências de passos temporais $t = 0, 1, 2, 3, 4, \ldots$, onde em cada passo t o ambiente envia ao agente uma representação de seu *estado*. Um agente, a cada passo, precisa decidir que ação executar, dentre as disponíveis, dado o estado do ambiente. A função que mapeia os estados do ambiente nas ações que o agente deve tomar é definida como a sua *política*. Uma decisão π_t em um estado de tempo t, oriunda de uma política π, define a probabilidade de se escolher uma ação a, dado um estado s. A política de um agente de aprendizado por reforço pode mudar ao longo do tempo devido a experiência que este vai adquirindo. Além disso, é objetivo do agente ser capaz de aumentar a quantidade de recompensa que recebe ao longo do tempo.

Episódios Em geral se deseja maximizar o *retorno esperado*, sendo G_t, definido como uma função a partir da sequência de recompensas. Em outras palavras $G_t = R_{t+1} + R_{t+2} + R_{t+3} + R_{t+4} + ... + R_T$, onde T é o ultimo passo temporal. Isso faz sentido ao considerarmos situações onde se tem noção de fim, em que a interação entre o ambiente e o agente pode ser separada em subsequências, chamadas de *episódios*. Durante o aprendizado do agente, consideramos que existe uma sequência de episódios, onde cada um consiste em uma sequência finita de passos temporais, sendo que numeramos cada passo temporal a partir do zero no início de cada episódio. Assim, nos referimos a um estado em um episódio não apenas como S_t mas sim como $S_{t,i}$ ou seja, esse é o estado S_t do episódio i.

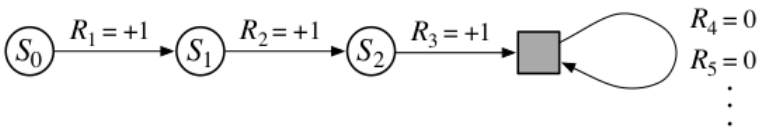

Figura 2. ***Autômato que representa o retorno esperado*** **[Sutton and Barto 1998]**

A propriedade de Markov Para descobrir a política a ser seguida no ambiente, o agente de Aprendizado assume que o ambiente é regido pela propriedade de Markov, e temos então um processo de decisão de Markov (MDP). Por simplicidade, vamos assumir que o número de estados e recompensas é finito. Tomando como exemplo um ambiente que responda no tempo $t+1$ a uma ação tomada no tempo t, no caso mais simples, essa resposta depende de tudo que ocorreu anteriormente. Para definir essa situação podemos utilizar a seguinte distribuição de probabilidade conjunta:

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

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

No entanto, se o estado do ambiente possui a propriedade de Markov, quando o ambiente responde no tempo $t+1$, a resposta depende apenas do evento anterior, em outras palavras, depende apenas do estado s e ação a no tempo t, conforme a equação 2:

$$p(s', r|s, a) = Pr\{S_{t+1} = s', R_{t+1} = r | S_t = s, A_t = a\} \tag{2}$$

Se o ambiente possui a propriedade Markov, podemos predizer qual o próximo estado do ambiente e a recompensa esperada, dado o estado atual do ambiente e a ação tomada pelo agente. Assim, podemos definir o conceito de Processo de Definição de Markov (MDP) que inclui: (i) Um conjunto finito de estados S; (ii) Um conjunto de ações disponíveis para cada estado A; (iii) Transições entre os estados, usualmente dispostas como uma distribuição de probabilidades ligando um estado $s \in S$ aos seus possíveis sucessores $s' \in S$; (iv) Uma função de recompensa $R(s, a), a \in A$ e $s \in S$; e (v) Um fator de desconto $0 \geq \gamma \leq 1$. Conforme dito anteriormente, um processo de decisão de Markov segue a premissa de que o futuro é independente do passado, dado o presente.

Q-Learning Dados os conceitos de política, episódios e processo de decisão de Markov, podemos apresentar o algoritmo de Aprendizado por Reforço utilizado neste artigo, a saber o algoritmo Q-leaning [Sutton and Barto 1998]. Em Aprendizado por Reforço, tentamos maximizar a soma das recompensas ao longo do tempo, ou seja:

$$\sum_{t=0}^{t=\infty} \gamma^t r_t(s, a) \tag{3}$$

Como o valor das recompensas futuras não é conhecido, podemos usar um algoritmo de iteração de valor para estimá-lo. Assim, o algoritmo $Q - learning$ tem como objetivo aprender uma estimativa do valor do melhor mapeamento entre estados e ações, usando uma abordagem livre de modelo que se baseia em uma função de valoração de ações chamada Q. A função Q recebe um estado e uma ação e retorna a recompensa futura esperada para aquela ação naquele estado. A ideia é que o valor retornado pela função Q cada vez se aproximará mais do valor real, dada a exploração no ambiente. A Equação 4 exibe como esse valor é calculado e atualizado.

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha^t \times (r(s_t, a_t) + \gamma^t \times \max_a Q(s_{t+1}, a) - Q(s_t, a_t)) \tag{4}$$

onde $s \in S$, $a \in a$, r é a recompensa e α é a taxa de aprendizado, que define o quanto queremos atualizar o valor corrente de Q (quando α é próximo de 1, o valor antigo é

simplesmente substituído pelo novo). Observe que ambas a recompensa atual ($r(s_t, a_t)$ e a recompensa futura estimada (o máximo valor esperado no estado sucessor, para cada ação, $\max_a Q(s_{t+1}, a)$) são levados em consideração ao se computar o valor da função Q. O valor encontrado pela da função é usado para determinar a ação ser executada em cada estado, usando a abordagem ϵ-*greedy*, em que, a cada iteração, com uma certa probabilidade é escolhida a melhor ação, ou uma ação aleatória, caso contrário. 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: Algoritmo Q-learning

Entrada: γ, α, ϵ
Inicialize $Q(s, a) \forall s, a, s \in S, a \in A$, de forma arbitrária
para cada *episódio* e **faça**
  $t \leftarrow 0$
  escolha um estado inicial aleatório
  enquanto *não atingir o objetivo* **faça**
    observe o estado s_t
    com uma probabilidade ϵ escolha a melhor ação a para s_t de acordo com $Q(s, a)$; caso contrário escolha uma ação aleatória a
    execute a no ambiente
    receba a recompensa $r(s, a)$
    observe o próximo estado s_{t+1}
    $\delta \leftarrow (r(s_t, a_t) + \gamma^t \times \max_a Q(s_{t+1}, a) - Q(s_t, a_t))$
    $Q(s, a) \leftarrow Q(s_t, a_t) + \alpha^t \delta$
    $t \leftarrow t + 1$
  fim
fim

2.2. *Workflows* Científios em Nuvens

Muitos experimentos científicos *in silico* são compostos pelo encadeamento de múltiplas combinações de programas que podem consumir grande quantidade de dados, e são comumente executados em ambientes de processamento de alto desempenho (PAD). Nesses experimentos, cada programa é executado consumindo um grupo específico de parâmetros e dados, cada qual com a sua própria semântica e sintaxe. A saída de um programa é normalmente utilizada como entrada para outro programa no encadeamento [Deelman et al. 2009, Mattoso et al. 2010]. *Workflows* são abstrações utilizadas para estruturar esse encadeamento de forma que possa ser executado e gerenciado por sistemas de *workflows*. O formalismo exposto nessa seção é baseado em [Oliveira et al. 2012].

Dado um *workflow W*, um conjunto *Ac* = ac_1, ac_2, ..., ac_k de ativações é criado para a execução paralela de *W*. Cada ativação ac_i está associada a uma atividade a_i específica que é representada como *Act*(ac_i) = a_i. O conjunto de ativações associadas a uma determinada atividade a_i do *workflow W* é denotada como *Ac*(a_i). Consideremos também $VM = \{VM_0, VM_1, ..., VM_{n-1}\}$ como o conjunto de *n* VMs criadas na nuvem e disponíveis para execução do *workflow*. Considere também $\varphi(W, VM)$ como sendo o escalonamento total das ativações pertencentes ao conjunto *AC* do *workflow W* em *VM*. Dado um *workflow W* que possui uma série de atividades $A = \{a_1, a_2, ..., a_n\}$ e um conjunto de ativações $AC = \{ac_1, ac_2, ..., ac_k\}$, seja $\varphi(W, VM) = \{sched_1, sched_2, ..., sched_k\}$ o conjunto dos escalonamentos para execução paralela do conjunto de ativações *AC*. O escalonamento de uma ativação é definido como *sched*($ac_i, VM_j, inicio, fim$), em que *início* é o tempo de

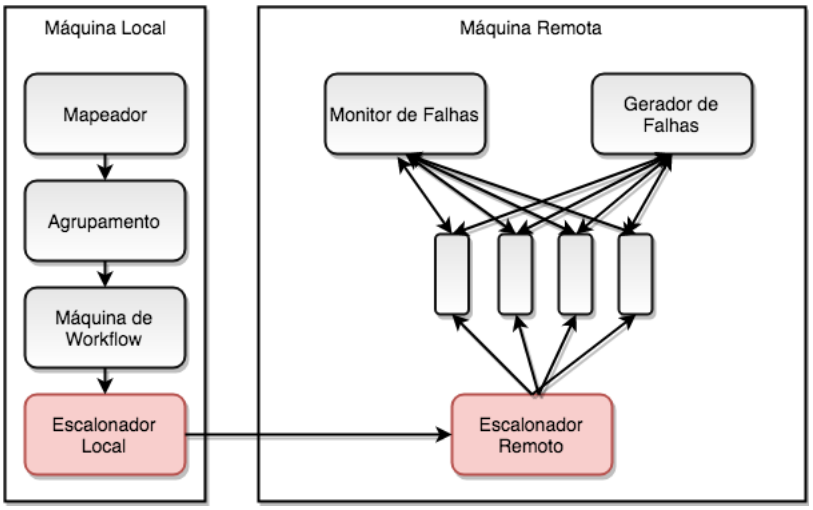

Figura 3. ***Arquitetura do WorkflowSim***

início de execução de uma ativação ac_i em uma dada VM_j e *fim* o tempo fim. Com o uso de algoritmos de aprendizado por reforço são definidos diversos escalonamentos *(sched)* ao longo da execução do *workflow*, a fim de 'aprender' qual o melhor escalonamento para o *workflow* e o ambiente.

3. Abordagem Proposta: $Workflow Sim4RL$

Nessa seção apresentamos como o $Workfllow Sim4RL$ foi estendido a partir do $Workflow Sim$ e como o mesmo realiza o escalonamento baseado em aprendizado por reforço.

3.1. O $Workflow Sim$

Segundo [Chen and Deelman 2012, Travassos and Barros 2003], a simulação é um dos métodos de avaliação mais populares em estudos baseados em *Workflows* Científicos, uma vez que a execução de testes reais em larga escala pode ser custosa tanto financeiramente quanto em relação ao tempo necessário. Dessa forma, uma vez que a proposta de algoritmos de aprendizado por reforço é executar diversas vezes um *Workflow* em um tempo hábil, para simular um ambiente com sobrecargas provenientes de um sistema heterogêneo e sujeito a falhas, optamos pela extensão de um simulador já amplamente conhecido, o *WorkflowSim* [Chen and Deelman 2012], justamente por fornecer um arcabouço capaz de lidar com tais questões. A arquitetura original do $Workflow Sim$ é apresentada na Figura 3, onde os componentes em vermelho representam as extensões propostas nesse artigo ($Workflow Sim4RL$).

O $Workflow Sim$ é composto por 7 componentes principais: (i) o mapeador que lê um arquivo de entrada e cria os objetos em memória com a estrutura do *workflow*, (ii) o componente de agrupamento que agrupa ativações para tornar o escalonamento mais eficiente, (iii) a máquina de *workflow* que controla as dependências e informa quais ativações podem executar, (iv) o escalonador local, que decide quis ativações são despachadas para o ambiente de processamento de alto desempenho, (v) o monitor de falhas que verifica a ocorrência de falhas nas máquinas remotas, (vi) o gerador de falhas que se encarrega de gerar falhas na execução, simulando um ambiente de nuvem real e (vii) o escalonador remoto que controla a execução das ativações nas máquinas remotas. Uma das grandes vantagens de se utilizar o $Workflow Sim$ é que o mesmo é capaz de simular questões como falhas e

flutuações de desempenho de forma bastante consistente e perto do ambiente real, fazendo com que ele seja uma ferramenta bastante apropriada para utilizarmos nesse artigo. A seguir, apresentamos como o $WorflowSim$ foi estendido para acomodar o algoritmo *Q-learning* para o escalonamento de ativações.

3.2. $WorkflowSim4RL$: Estendendo o $WorkflowSim$

Inicialmente, para que seja possível utilizar a abordagem de Aprendizado por Reforço no $WorkflowSim$ temos que verificar quais componentes devem ser estendidos. Os únicos componentes que necessitam extensões são o escalonador local e o escalonador remoto, uma vez que são esses componentes que decidem qual máquina executará quais ativações. Dessa forma, antes de se estender os escalonadores, é necessário definir uma forma de avaliar o ambiente. As máquinas virtuais são os recursos do ambiente, logo, o desempenho de cada uma delas é avaliado individualmente após executar uma ativação, bem como globalmente, considerando todo o conjunto de máquinas virtuais da nuvem. Isso é necessário para que possamos calcular o valor de recompensa/punição a cada decisão tomada pelo agente de Aprendizado por Reforço (*i.e.* o escalonador). Para tanto, foram definidas funções para calcular o desempenho de cada máquina virtual ao executar as ativações do *workflow*. Como métrica, foram utilizados o tempo de execução e o tempo de fila de espera.

Inspirados na função de desempenho proposta por [Costa et al. 2009] para ambientes de grade, definimos o desempenho de uma máquina virtual de acordo com o tempo de execução de uma ativação ac_i na máquina (te) em relação ao tempo de espera em fila (tf) para ser executada. Considerando tt (tempo total) como $tt = te + tf$, tais valores são usados para calcular o índice de desempenho (Pi) de uma máquina virtual ao executar uma ativação, Pi = $te_i * \alpha$ + (1 – α) * (tf). No caso do desempenho global (todas as máquinas virtuais envolvidas na execução do *workflow*, já que a solução gerada deve ser boa para máquinas individuais, mas também o deve ser de forma global) $\overline{Pi} = \overline{te}$ * α + (1 – α) * $(\overline{tf})$, sendo $\overline{te}$ e $\overline{tf}$ as médias dos tempos de execução e tempo de fila respectivamente. Além do desempenho individual da máquina e o desempenho global em uma execução, calculamos também o desempenho acumulado de cada máquina virtual, *i.e.* para calcular o desempenho de todas as execuções de uma determinada máquina vm_i, como $\overline{Pi_i}$ = $\overline{te_i} * \alpha$ + (1 – α) * $(\overline{tf_i})$, onde $\overline{te_i}$ e $\overline{tf_i}$ são as médias dos tempos de execução e de fila de todas as execuções da máquina virtual respectivamente.

Foi definida também a função Pi que retorna um valor numérico que representa um índice de desempenho da máquina virtual ao executar uma ativação, dado o tempo em que tal ativação permaneceu na fila de espera da máquina para execução. Em suma, Pi representa o quão rápido uma máquina virtual foi capaz de executar uma ativação em relação ao tempo em que a ativação ficou sem ser executada.Desta forma, somos capazes de calcular o desvio padrão $(stdv)$ de $\overline{Pi}$, e avaliamos a alocação da ativação para ser executada na máquina da seguinte forma: r = -1 se $Pi > (Pim + stdv)$ (é aplicada uma punição) e r = 1 se $Pi < (Pim - stdv)$ (é aplicada uma recompensa);

Por fim, calculamos a eficiência de uma máquina virtual atribuindo um valor chamado de $(nrle)$ (nova eficiência) definido como $nrle = rle + \alpha * (r - rle)$, sendo rle o valor prévio de eficiência da máquina e α o parâmetro associado ao *Q-Learning* para definir a velocidade de aprendizado. O valor retornado pela equação de eficiência $(nrle)$ nos dá a recompensa ou punição que são dados ao agente por uma escolha boa ou ruim de escalonamento, a serem utilizados no Algoritmo Q-Learning implementado neste artigo. De forma a se avaliar as ações do escalonador com aprendizado por reforço, foi necessário um mapeamento de todos os possíveis estados do ambiente em que o mesmo se encontra. Para isso, consideramos

que cada estado de uma tarefa dentro do grafo de dependências do *workflow* combinado com cada estado de uma máquina virtual formam um estado do próprio grafo, em outras palavras, o estado do grafo do *workflow* depende do estado da máquina e da ativação. A seguir apresentamos todos os possíveis estados do ambiente de nuvem, assumindo *s* como estados das ativações e *S* o conjunto de estados das ativações do *workflow*. Essa tabela é gerada e salva em arquivo após o término de cada simulação utilizando o agente de Aprendizado por Reforço de escalonador proposto neste artigo.

Definição dos Elementos do MDP para o Escalonamento em Workflows. Para aplicar o algoritmo $Q - Learning$ que assume que o ambiente se comporta como um MDP, é necessário definir os estados, ações e o modelo de transição. Considerando o conjunto M que define os estados dos recursos, ou seja, das máquinas disponíveis, temos que $M = \{idle, running\}$. Por outro lado, a definição do conjunto de estados referente ao Workflow requer mais cuidado, uma vez que este depende dos estados das ativações que o compõem. Assim, definimos o conjunto $S = \{ready, running,$ finished-with-failure, finished-with-success$\}$ como sendo o conjunto de estados em que o workflow pode se encontrar em um determinado instante de tempo. As ativações do workflow, por sua vez, podem estar em um dos estados pertencentes ao conjunto $S' = \{ready, locked, running,$ finished-with-success, finished-with-failure$\}$. Assim, os estados do workflow são definidos a partir dos estados de suas ativações, conforme abaixo:

- $s = ready \in S$ se $\exists$ uma tarefa com $s' = ready \in S'$
- $s =$ finished-with-success $\in S$ se $\forall s' \in S', s' =$ finished-with-success
- $s =$ finished-with-failure $\in S$ se $\exists s' \in S', s' =$ finished-with-failure
- $s = running \in S$ se $\nexists s' \in S' \mid s' = ready$

Vale ressaltar que o objetivo do Aprendizado por Reforço é que o workflow alcance o estado de $finished - with - success$. Como as ações envolvem alocar uma ativação do Workflow em um máquina, temos o conjunto de ações $A = \{\{act_i\ to\ m_j\}, end\}$, onde act_i é uma ativação do Workflow e m_j é uma máquina, $act_i\ to\ m_j$ representa a alocação de uma ativação i para uma máquina j, e end significa encerrar a execução do Workflow. A transição de estados do Workflow encontra-se descrita na Tabela 1.

Tabela 1. Tabela de Transição de Estados do Workflow

Máquina	Estado	Próximo Estado	Ação
idle	ready	ready	act_i to m_j
idle	ready	ready	-
idle	ready	running	act_i to m_j
idle	running	ready	-
idle	running	running	-
idle	running	finished-with-success	end
idle	running	finished-with-failure	end
running	ready	ready	-
running	running	ready	-
running	running	running	-
running	running	finished-with-success	end
running	running	finished-with-failure	end

Consideramos que cada execução do *workflow* representa um episódio para o algoritmo *Q-Learning*. Desta forma, ao término de uma simulação, a tabela de estados de cada máquina virtual é salva em arquivo, também é atualizada para cada par *(estado, ação)*, o valor da função de avaliação Q do *Q-Learning*. Além disso, o arquivo também contém o plano de escalonamento gerado naquele episódio, o tempo total de simulação até o fim do episódio correspondente ao arquivo, e o tempo simulado do *workflow* naquele episódio, a fim de auxiliar as avaliações realizadas a seguir.

Conforme discutido anteriormente, o *Q-Learning* depende da progressão dos episódios ao longo do tempo, tomando cada execução do simulador como um episódio. Assim, cada simulação é alimentada com dados da execução imediatamente anterior. Entretanto, isso gera um desafio no caso do primeiro episódio. Dados como o tempo de execução de uma ativação precisam ser fornecidos no primeiro episódio, afim de que seja possível calcular o índice de desempenho Pi, para que seja possível calcular o novo valor da função Q do *Q-Learning*. Como os métodos apresentados anteriormente estão em função do tempo de execução e tempo de espera em fila, esses dados de tempo precisam ser coletados antes do início de uma simulação. Assim, antes de iniciar o primeiro episódio, cada ativação foi executada individualmente a fim de coletarmos o tempo de execução simulado da ativação gerado pelo $WorkflowSim4RL$.

O escalonador *RFLSchedulingAlgorithm* implementa a a utilização do escalonador aprendido, bem como o algoritmo Q-learning usado para aprendê-lo. O *RFLSchedulingAlgorithm* recebe uma lista de ativações ainda não escalonadas, e de máquinas virtuais disponíveis. As máquinas virtuais são escolhidas de acordo com o estado do grafo de dependências do *workflow* e o valor da função de avaliação Q do algoritmo *Q-Learning*, presente no arquivo gerado no episódio anterior.

Escolhida a máquina virtual para uma determinada ativação, é feita a alocação da ativação para ser executada, o valor de tempo de fila é calculado conforme mencionado anteriormente, o novo estado do *workflow* é calculado, e então o valor da função de avaliação Q é calculado para esse novo estado. Ao término desse processo, o simulador inicia a simulação normalmente, e ao término dela os dados obtidos no episódio atual são persistidos para serem usados no episódio seguinte.

4. Avaliação Experimental

De forma a avaliar o escalonamento com Aprendizado por Reforço, executamos uma série de experimentos simulados e um experimento real para avaliar a qualidade do plano de escalonamento aprendido. Para avaliar o $WorkflowSim4RL$ utilizamos um conjunto variado de *traces* de *workflows* disponibilizados pela equipe do SGWfC Pegasus[1], a saber: (i) **Montage**: *workflow* criado pela NASA que cria um mosaico de várias imagens de entrada para criar uma foto personalizada do céu; (ii) **Cybershake**: *workflow* usado pelo Centro de Prevenções de Terremotos da Calfornia para caracterizar riscos de terremoto na região; (iii) **Epigenomics**: *workflow* que automatiza tarefas de mapeamento do genoma; (iv) **LIGO Inspiral**: *workflow* usado para gerar e analisar formas de onda gravitacionais de dados coletados durante o coalescência de sistemas binários compactos; e (v) **SIPHT**: *workflow* usado para automatizar a pesquisa de RNAs não traduzidos (sRNAs) para bactérias no banco de dados NCBI. Cada um desses *workflows* apresenta diferentes características, como ativações intensivas em CPU ou ativações intensivas em dados, por exemplo.

Além de variar o *workflow*, variamos a quantidade de ativações que cada *workflow*

[1]https://confluence.pegasus.isi.edu/display/pegasus/WorkflowGenerator

executou, a saber: Montage (25, 50, 100 e 1000 ativações), Cybershake (30, 50, 100 e 1000 ativações), Epigenomics (24, 46, 100 e 997 ativações), LIGO Inspiral (30, 50, 100 e 1000 ativações) e SIPHT (30, 60, 100 e 1000 ativações) conforme apresentado na Tabela 3. Além disso, cada *workflow* foi escalonado em 5 conjuntos distintos de máquinas virtuais (Tabela 2). Em cada execução, foram gerados 100 episódios. Cada conjunto de máquina foi elaborado de modo que se obtenham 8, 16, 32, 64, e 128 CPUs virtualizadas.

Foram simulados dois tipos de máquinas dentro do $WorkflowSim4RL$ seguindo o padrão da Amazon AWS, a *m3.medium* com 1 CPU virtualizada, e a *m3.2xlarge* com 8 CPUs virtualizadas. Em termos de memória RAM, a máquina *m3.medium* possui 3,75 GB e a máquina *m3.2xlarge* possui 30 GB. Um fator importante para o $WorkflowSim4RL$ é o valor de micro instruções por segundo (MIPS). O parâmetro MIPS no $WorkflowSim4RL$ é que de fato define a heterogeneidade do ambiente. Definimos o valor de MIPS para a máquina *m3.medium* como 1.000 e para a máquina *m3.2xlarge* 4.156 conforme definido pela Amazon[2].

Tabela 2. Conjuntos de máquinas

8 vCPUs	16 vCPUs	32 vCPUs	64 vCPUs	128 vCPUs
-	8 - *m3.medium*	8 - *m3.medium*	8 - *m3.medium*	8 - *m3.medium*
1 - *m3.2xlarge*	1 - *m3.2xlarge*	3 - *m3.2xlarge*	7 - *m3.2xlarge*	15 - *m3.2xlarge*

Em cada uma das execuções, o *Q-Learning* e a tabela com os valores de $Q(s, a)$ são inicializadas com o valor de $0,2$ e aplicamos um fator de aleatoriedade na escolha da máquina virtual a ser usada para a alocação das ativações. Desta forma, o escalonador escolhe a máquina virtual por meio de um fator probabilístico, ou o agente escolhe a máquina que possuir o maior valor da função de avaliação Q que se encontra na tabela presente no arquivo do episódio anterior, seguindo a abordagem ϵ-*greedy*. Além disso, o parâmetro de peso α foi definido na função Pi como $0,5$, o valor de γ do algoritmo *Q-Learning* como $0,5$ e o fator de velocidade de aprendizado l como igual a $1,0$. Os tempos de execução em segundos de cada um dos *workflows* no ambiente definido são apresentados na Tabela 3. Podemos verificar que as execuções com mais vCPUs apresentaram um tempo de execução reduzido (conforme esperado), porém, em muitos casos, o valor do *speedup* não foi alto, em especial para o *workflow* Epigenomics, que apresentou as menores reduções de tempo de execução ao se adicionar mais máquinas virtuais.

Além da avaliação do tempo de execução simulado, executamos um dos planos de escalonamento gerado pelo $WorkflowSim4RL$ em um experimento real utilizando o SGWfC SciCumulus. O workflow escolhido foi o Montage com 25 ativações associadas. Comparamos o tempo de execução do workflow escalonado utilizando aprendizado por reforço com o escalonamento FCFS *First Come First Served*. Conforme apresentado na Figura 4, o escalonamento com aprendizado por reforço apresentou um tempo de execução menor que o FCFS em todas as combinações de máquinas virtuais analisadas. Apesar de novos experimentos mais complexos se mostrarem necessários, os resultados aqui apresentados mostram a viabilidade do $WorkflowSim4RL$.

5. Trabalhos Relacionados

Em [Barrett et al. 2011] foi proposta a utilização de um algoritmo genético para a evolução do aprendizado do agente, utilizando MDP para a escolha dos melhores planos de escalonamento

[2]www.cmips.net/2014/04/01/amazon-40-cut-and-cmips-benchmarks-with-amazon-m3-2xlarge-and-c3-4xlarge/

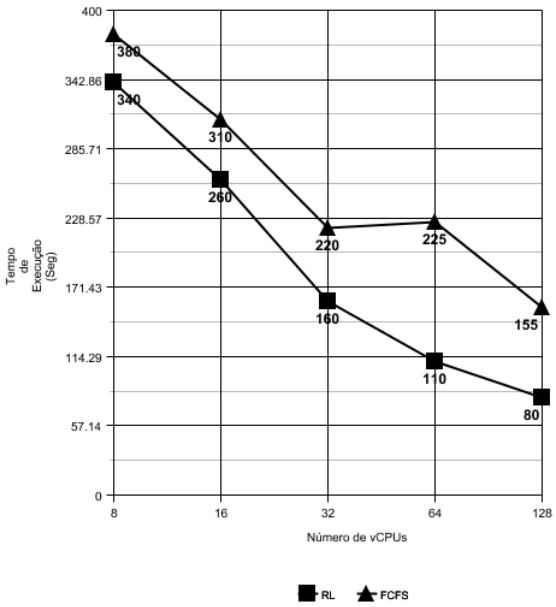

Figura 4. ***Tempo de execução do Montage_25 executado com o SciCumulus***

para *Workflows* em conjunto com um modelo Bayesiano. Apesar de ter objetivo semelhante a abordagem proposta nesse artigo (*i.e.*, reduzir o tempo de execução do *Workflow*), a abordagem proposta por [Barrett et al. 2011] foca também em reduzir custos financeiros. Nossa abordagem também se diferencia por utilizar um escalonador que implementa o *Q-Learning* no simulador *WorkflowSim*, evitando que o tempo de aprendizado do agente seja contabilizado na execução do *workflow* de fato.

Apesar da abordagem proposta nesse artigo se inspirar no trabalho proposto por [Costa et al. 2009], utilizando a ideia de funções de desempenho por recurso e a definição do valor de recompensa/punição, a abordagem de [Costa et al. 2009] é focada no ambiente em Grades Computacionais. Além disso, nossa abordagem se difere na implementação do algoritmo de aprendizado por reforço *Q-Learning* juntamente com as métricas geradas pelas funções de desempenho propostas por [Costa et al. 2009].

6. Conclusão e Trabalhos Futuros

Neste artigo, apresentamos uma abordagem para escalonamento de *workflows* científicos em nuvens de computadores, a qual utilizou Aprendizado por Reforço como técnica para a descoberta de um escalonador das máquinas virtuais disponíveis para execução. A abordagem proposta é uma alternativa aos escalonadores existentes nos SGWfCs para nuvem, pois não necessita que características do ambiente sejam formalizadas de antemão. Ao mesmo tempo, a abordagem proposta é de simples implementação, necessitando somente ser capaz de avaliar se um determinado plano de escalonamento teve um bom resultado ou não. Nesse trabalho, estendemos o $WorkflowSim$, criando assim o $WorkflowSim4RL$ que implementa a abordagem proposta nesse artigo. Os resultados experimentais mostram que o simulador baseado em aprendizado por reforço é capaz de se beneficiar do ambiente heterogêneo, gerando escalonamentos com tempos menores a medida que a quantidade de recursos aumenta.

Além disso, realizamos uma execução real do *workflow* Montage com 25 ativações utilizando o plano de escalonamento da abordagem proposta e o gerado pelo FCFS

Tabela 3. Tempo de Execução dos Workflows no WorkflowSim4RL (em segundos)

	Número de Núcleos Virtuais			
Workflows	**16**	**32**	**64**	**128**
CyberShake_30	6.559,71	6.517,74	6.278,79	6.166,39
CyberShake_50	12.762,63	12.550,50	12.316,62	12.183,19
CyberShake_100	25.354,15	25.113,91	24.562,39	24.199,95
CyberShake_1000	47.807,61	44.118,30	40.629,30	38.257,61
Epigenomics_24	15.085,01	9.496,22	9.798,79	7.766,11
Epigenomics_46	33.232,34	31.314,53	22.698,94	17.552,66
Epigenomics_100	320.684,21	254.145,95	197.687,41	201.803,33
Epigenomics_997	3.006.902,18	2.246.804,08	1.707.749,73	1.422.339,80
Inspiral_30	6.176,08	4.454,18	4.586,26	3.374,25
Inspiral_50	9.720,62	8.972,35	7.286,11	6.228,70
Inspiral_100	16.078,31	12.785,12	10.975,93	10.184,21
Inspiral_1000	172.362,68	126.508,68	95.578,53	77.739,38
Montage_25	226,70	174,01	175,80	114,62
Montage_50	477,42	424,33	330,19	250,10
Montage_100	950,57	774,25	647,64	520,82
Montage_1000	9.938,84	7.834,66	6.347,65	5.264,15
Sipht_30	3.733,51	5.180,81	5.213,3	1.388,70
Sipht_60	9.643,12	7.425,38	6.878,84	2.968,70
Sipht_100	15.379,13	14.702,93	10.397,34	5.846,22
Sipht_1000	131.305,53	102.202,37	75.976,43	66.467,61

(padrão do $WorkflowSim$). Verificamos que a abordagem proposta apresentou um tempo de execução menor para todas as combinações de máquinas exploradas. Dessa forma, verificamos que em ambientes dinâmicos e heterogêneos como as nuvens de computadores, estratégias adaptativas como aprendizado por reforço são importantes, mas acreditamos serem necessários mais experimentos com *workflows* reais e em maior escala para analisarmos o impacto do escalonamento com aprendizado por reforço em uma variedade de perfis de *workflows*. Como trabalhos futuros, poderíamos intercalar execuções simuladas no $WorkflowSim4RL$ com execuções reais no SciCumulus, de forma que a tabela do *Q-Learning* pudesse ser realimentada de acordo com os tempo reais, pois o tempo simulado pode ser ligeiramente diferente do tempo de execução real.

Referências

Barrett, E., Howley, E., and Duggan, J. (2011). A learning architecture for scheduling workflow applications in the cloud. In *Web Services (ECOWS), 2011 Ninth IEEE European Conference on*, pages 83–90. IEEE.

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

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

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

Deelman, E., Mehta, G., Singh, G., Su, M.-H., and Vahi, K. (2007). Pegasus: mapping large-scale workflows to distributed resources. In *Workflows for e-Science*, pages 376–394. Springer.

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

Mattoso, M., Werner, C., Travassos, G. H., Braganholo, V., Ogasawara, E., Oliveira, D., Cruz, S., Martinho, W., and Murta, L. (2010). Towards supporting the life cycle of large scale scientific experiments. *International Journal of Business Process Integration and Management*, 5(1):79–92.

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

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

Sutton, R. S. and Barto, A. G. (1998). *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge.

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

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

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