Algoritmo de Escalonamento baseado em *Deep Reinforcement Learning* para Computação em Grids

Lucas Casagrande, Charles Miers, Guilherme Koslovski, Maurício Pillon

[1]Programa de Pós-Graduação em Computação Aplicada (PPGCA)
Universidade do Estado de Santa Catarina – UDESC – Joinville – SC – Brasil

lucas.casagrande@edu.udesc.br, {charles.miers,guilherme.koslovski,
mauricio.pillon}@udesc.br

Abstract. *Scheduling algorithms play a key role in the optimization of resource utilization in grid computing. In this paper, a scheduling algorithm based on Deep Reinforcement Learning is proposed to minimize the slowdown, a metric that represents the quality of service. The algorithm is trained in a simulated environment and compared with traditional heuristic methods under different workload profiles. The results show that the algorithm achieves a reduction of up to 54% in the slowdown with a minor impact in the makespan. It is possible to conclude that DRL methods can learn scheduling policies that adapt to different workloads, being a viable option for the scheduling problem.*

Resumo. *Algoritmos de escalonamento desempenham um papel chave durante a otimização dos recursos de uma infraestrutura em grids. Neste trabalho, um algoritmo de escalonamento baseado em Deep Reinforcement Learning é proposto com o objetivo de minimizar o slowdown, uma métrica que representa a qualidade de serviço. O algoritmo é treinado via simulação e comparado com heurísticas tradicionais sob diferentes perfis de carga de trabalho. Resultados demonstram que o algoritmo apresenta uma redução de até 54% no slowdown, tendo um leve impacto no makespan. É possível concluir que métodos de DRL são capazes de aprender políticas de escalonamento que se adaptam a carga de trabalho, sendo uma opção viável ao problema de escalonamento.*

1. Introdução

Através do compartilhamento de recursos entre instituições geograficamente distribuídas, computação em grids consegue prover uma quantidade massiva de poder computacional sob demanda, se tornando uma opção favorável à computação de alto desempenho e paralela [Foster et al. 2008]. Neste contexto, *middlewares* chamados de *Resource and Jobs Management Systems* (RJMS) são utilizados no gerenciamento de uma infraestrutura em grids com a finalidade de maximizar a utilização e coordenar o compartilhamento dos recursos entre as demandas de cada usuário.

Dentre as principais atribuições de um RJMS, determinar o tempo e os recursos que devem ser alocados para cada uma das requisições recebidas é um dos processos chave durante a otimização de sua eficiência [Poquet 2017]. Este tipo de mapeamento, entre a demanda e os recursos disponíveis, também é conhecido como o problema de escalonamento sendo, a sua otimização, um problema da classe NP-Difícil. Dada essa complexidade, métodos heurísticos são, de um modo geral, a principal escolha na composição de diferentes políticas de escalonamento [Pooranian et al. 2015].

Métodos heurísticos funcionam bem no espaço de solução em que foram projetados e são previsíveis. Contudo, o seu desempenho pode ser negativamente influenciado pelo perfil de carga de trabalho que é submetida no sistema [Vasile et al. 2018]. Uma análise dos modelos de cargas de trabalho de ambientes em grids demonstra que as requisições chegam em rajadas, dando indícios que o sistema passa de tempos em espera para tempos com alta carga de *jobs* na fila [Li and Buyya 2009]. Em função desta natureza dinâmica, a adequação de uma heurística geral que se adapte em tais condições é um processo complexo de difícil execução. Nesta circunstância, algoritmos de escalonamento adaptativos são uma possível alternativa [Orhean et al. 2018].

No contexto de controle adaptativo, métodos de *Reinforcement Learning* (RL) introduzem uma abordagem computacional onde um agente consegue aprender uma política através da interação com um ambiente, seguindo uma abordagem de tentativa e erro. A cada interação, o agente recebe um sinal de recompensa que representa o custo da ação tomada na situação apresentada. A ideia consiste em reforçar as melhores ações com recompensas positivas de modo a aumentar a tendência com que elas sejam repetidas futuramente [Sutton et al. 1998].

Métodos comuns de RL, como Q-learning e SARSA, podem aprender uma política de escalonamento que se adapte a diferentes perfis de cargas de trabalho. Contudo, na medida com que a quantidade de estados e ações aumenta, tais métodos sofrem da maldição da dimensionalidade sendo impraticáveis em problemas com um conjunto grande de estados [Orhean et al. 2018]. Em contrapartida, métodos baseados em *Deep Reinforcement Learning* (DRL) minimizam este problema utilizando *Deep Neural Network*s (DNNs) na aproximação direta de uma política ou dos valores de cada par estado-ação. Ao eliminar a dependência decorrida da utilização de tabelas, DRL permite que problemas de RL com um grande conjunto de estados e ações, tanto discretos como contínuos, sejam resolvidos de modo eficiente [Mnih et al. 2015].

Deste modo, o presente trabalho propõe o treinamento de um agente utilizando o método *Proximal Policy Optimization* (PPO), da família dos métodos de gradiente de política, para aprender uma política de escalonamento com o objetivo de minimizar o slowdown, uma métrica que representa a qualidade de serviço. O método PPO demonstrou desempenho superior quando comparado com outras técnicas de DRL em ambientes como o MuJoCo e o *Arcade Learning Environment* (ALE) [Schulman et al. 2017]. Tal desempenho nos motiva a estender este método para um ambiente que simula uma infraestrutura em grids de modo a analisar o seu comportamento durante a solução do problema de escalonamento.

O artigo está organizado em cinco seções. Na segunda seção são apresentados alguns trabalhos relacionados. Na terceira seção é descrito o modelo do sistema, incluindo a plataforma da grid, a formulação do problema e a definição do método utilizado. Na quarta seção, a configuração dos experimentos e os resultados são apresentados e discutidos. Na quinta e última seção, são descritas as conclusões obtidas durante a condução deste trabalho.

2. Trabalhos Relacionados

O problema de escalonamento em grids é um tópico vasto que continua ativo na comunidade científica. O *Deadline-Based Backfilling* (DBF) é uma proposta que visa distinguir

os *jobs* que devem terminar o mais rápido possível daqueles que podem ser atrasados para eventualmente reduzir o tempo médio de espera [N'takpé and Suter 2018]. O uso de simulações para refinar heurísticas através da reordenação da fila de *jobs* também é aplicado com este mesmo objetivo [Lelong et al. 2018].

A degradação no desempenho de heurísticas em função do perfil da carga de trabalho é um problema conhecido e alguns trabalhos tem proposto a utilização de RL. Dentre as técnicas utilizadas está o uso do Q-Learning no desenvolvimento de um escalonador adaptativo para minimizar o makespan e o custo de comunicação decorrente da manipulação de dados entre *jobs* paralelos [Moghadam and Babamir 2018]. De modo semelhante, Q-Learning também é aplicado para minimizar o tempo de resposta médio [Tong et al. 2014]. Como um meio para aumentar a escalabilidade, esquemas descentralizados com múltiplos agentes independentes também são utilizados em conjunto com um mecanismo de gossip para disseminação do aprendizado [Wu and Xu 2018].

Como alternativa aos métodos de RL, alguns trabalhos também aplicam DRL tanto no escalonamento de *jobs* em *clusters* como na computação em nuvem. O DeepRM utiliza o método REINFORCE, da família dos métodos de gradiente de política, em conjunto com uma DNN para treinar um algoritmo capaz de minimizar o slowdown [Mao et al. 2016]. O DRL-Cloud é baseado na utilização de *Deep Q-Network*s (DQNs) para maximizar a eficiência energética durante o provisionamento de recursos e o escalonamento de tarefas em um ambiente em nuvem [Cheng et al. 2018]. Seguindo um esquema hierárquico, RL e DRL também são combinados com outras técnicas de *Machine Learning* (ML) para realizar a alocação de recursos e o gerenciamento de energia [Liu et al. 2017].

3. Modelo do Sistema

Nesta seção são apresentados os aspectos do modelo do sistema considerado no decorrer deste trabalho. São definidos a plataforma, composta pelo modelo dos recursos e dos *jobs*, a transformação do problema de escalonamento para um *Markov Decision Process* (MDP), necessário para a aplicação de métodos baseados em RL, e descrito o processo de treinamento do agente. Este modelo é definido com base em trabalhos relacionados [Mao et al. 2016, N'takpé and Suter 2018].

3.1. Plataforma

Uma plataforma P em grid é formada por recursos geograficamente distribuídos organizados em locais. Cada local possui uma quantidade variável de *clusters* que podem conter tanto máquinas homogêneas como heterogêneas. Cada máquina contém ao menos um processador com um ou mais núcleos n com capacidade de processamento medida em *Floating-point Operations Per Second* (FLOPS). Neste trabalho, a comunicação entre os *clusters* e locais é desconsiderada. Portanto, a plataforma pode ser vista como um *pool* de recursos definida pela capacidade de processamento dos núcleos dos processadores em cada uma de suas máquinas $P = \{n_1, n_2, ..., n_i\}$.

Jobs j chegam de modo online, a qualquer momento, e são caracterizados de acordo com o seu tempo de submissão sub_j, a quantidade de recursos res_j requisitada e uma estimação do tempo $wall_j$ máximo em que utilizarão os recursos. Ambas as quantidades de recursos res_j e de tempo $wall_j$ são definidas pelo usuário e desconhecidas pelo sistema até o momento de sua submissão.

Por simplicidade, consideramos que o tempo de execução p_j dos *jobs* é conhecido e coincide com o tempo $wall_j$ definido pelo usuário. Também é assumido que os recursos são homogêneos, com a mesma capacidade computacional, e cada *job* pode requisitar uma quantidade finita de núcleos $res_j \in [1, |N|]$, que, daqui em diante, chamaremos apenas de recursos. Portanto, um *job* pode ser definido como $j = \{sub_j, res_j, wall_j\}$, onde res_j também determina a quantidade de tarefas que são executadas em paralelo.

É considerado que os *jobs* são rígidos e preemptividade não é permitida. Portanto, um recurso somente pode ser liberado em duas situações: após o término de todas as tarefas do *job* que possui este recurso; ou quando o $wall_j$ expira e o *job* é automaticamente removido do sistema. O escalonador não é responsabilizado pelos *jobs* que expiraram o seu tempo.

3.2. Formulação do Problema

Para permitir a aplicabilidade de algoritmos de RL, o problema é formulado como sendo um MDP de horizonte finito formado pela tupla $\{S, A, T, R\}$, onde S é o conjunto de estados em que o ambiente pode estar em um determinado momento, A determina o conjunto de ações possíveis, T determina a dinâmica de transição entre os estados, considerada como sendo desconhecida, e R é uma função de recompensa que retorna um valor representando o custo da ação tomada. O tamanho do horizonte é limitado conforme a quantidade de *jobs* da carga de trabalho submetida no sistema.

Conjunto de Estados: O estado do sistema é representado por imagens distintas que definem tanto o estado dos recursos como as características de cada *job* na fila. Na Figura 1 é demonstrada uma representação do estado contendo 5 recursos, 3 espaços para os *jobs* e um espaço que indica a quantidade restante de *jobs* na fila.

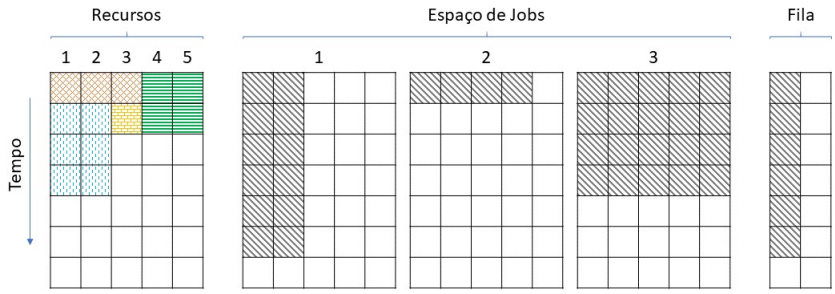

Figura 1. Representação de um estado do ambiente

Com base na Figura 1, cada espaço de *job* representa a quantidade de recursos res_j, no eixo x, e o tempo $wall_j$, no eixo y, requisitado por um *job* j. Como exemplo, o *job* no primeiro espaço requisitou 2 recursos por 6 unidades de tempo enquanto o *job* no segundo espaço requisitou 4 recursos por 1 unidade de tempo. Demais *jobs* seguem esta mesma lógica e, na medida com que são consumidos pelo escalonador, *jobs* da fila são retirados para preencher estes espaços respeitando a sua ordem de chegada.

Em relação ao espaço de recursos, este representa o tempo em que cada *job*, diferenciado por uma cor distinta, irá ocupar os recursos alocados ou reservados. A ideia é semelhante a de um diagrama de Gantt. Na Figura 1, o espaço de recursos contém 4 *jobs* sendo que o primeiro está utilizando 3 recursos e deve liberá-los na próxima unidade de tempo enquanto o segundo utiliza apenas 2 recursos com um tempo restante de duas

unidades. Demais *jobs* representados neste espaço estão reservados para iniciar a sua execução imediatamente após a liberação dos recursos que lhes foram reservados.

Conjunto de Ações: O conjunto de ações é determinado pela quantidade de espaços de *jobs* que estão representados no estado. Em qualquer momento o agente pode escolher tanto avançar uma unidade de tempo ou selecionar um *job* para ser escalonado nos primeiros recursos disponíveis. Logo, o conjunto de ações é definido por $A = \{\emptyset, 1, 2, ...M\}$, onde M representa a quantidade de espaços de *jobs* e $\emptyset$ é uma ação vazia que corresponde a opção de pular uma decisão.

Ações inválidas, quando o agente seleciona um *job* que não pode ser escalonado devido a falta de recursos, são interpretadas como sendo uma ação vazia. Enquanto o agente seleciona apenas ações válidas, o tempo é parado para permitir que múltiplas decisões possam ser tomadas em uma única unidade de tempo. Somente quando uma ação vazia é tomada, tanto de modo explícito como implícito, é que o tempo avança por uma unidade de tempo.

Função de Recompensa: O objetivo do agente é aprender uma política que maximize a sua recompensa esperada através da sucessiva tomada de decisões. Com o objetivo de minimizar o slowdown, a função de recompensa é formulada como sendo a sua média negativa. Com esta finalidade, o slowdown é estimado por $\Sigma_{j \in J} \frac{1}{p_j}$, onde p_j representa o tempo de execução do *job* j, 1 corresponde ao avanço de uma unidade de tempo e J representa todos os *jobs* que estão no sistema. Logo, a função de recompensa pode ser interpretada como sendo uma penalização proporcional a média do slowdown.

3.3. Método de Aprendizado

O método PPO representa um avanço na área de RL no que diz respeito a métodos baseados no gradiente de política. Tais métodos buscam aprender uma política parametrizada, aumentando ou diminuindo a probabilidade de suas ações com base na recompensa recebida em cada estado. Este comportamento o distingue dos métodos baseados em valores, como Q-Learning e DQN, que baseiam a sua tomada de decisões utilizando uma estimação do valor de cada par estado-ação [Sutton et al. 1998].

De um modo geral, PPO difere dos demais introduzindo uma função objetivo que permite a execução de múltiplas épocas de treinamento. Tal função é dada por $L(\theta) = \hat{\mathbb{E}}_t \Big[min(r_t(\theta)\hat{A}_t, clip(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t) \Big]$, onde $\hat{A}_t$ representa a *advantage function*, $r_t(\theta)$ retorna a razão entre a política que está sendo otimizada e a anterior, utilizada durante a interação com o ambiente, e o parâmetro ϵ introduz um limite nos valores possíveis de $r_t(\theta)$. Ao limitar $r_t(\theta)$, atualizações agressivas na política são desincentivadas, evitando que a política sendo otimizada se distancie muito em relação a política anterior. Logo, isto torna possível a reutilização dos dados durante a otimização sem que a política seja destruída, aumentando a sua eficiência.

Sobre o seu modo de funcionamento, o algoritmo segue o estilo dos métodos *actor-critic*. O algoritmo inicia com uma política aleatória e utiliza um número arbitrário de *actors* para interagir com o ambiente paralelamente. Após um número fixo de interações, o conjunto de estados, recompensas e ações, coletado pelos *actors*, é utilizado em conjunto com a função valor, estimada pelo *critic*, para otimizar a política por um número arbitrário de épocas. Após atualizar a política, novas informações são coletadas

pelos *actors* e todo o processo é repetido até que um momento de parada seja alcançado [Schulman et al. 2017].

4. Experimentação

4.1. Configuração

Para coletar resultados, os experimentos são simulados utilizando o Batsim. Este simulador utiliza o Simgrid para simular uma infraestrutura em grids e apresenta resultados próximos de um ambiente em produção [Poquet 2017]. Seguindo a sua estrutura, a plataforma é formada por 10 recursos homogêneos, com a mesma capacidade de FLOPS, enquanto a carga de trabalho é composta por *jobs* paralelos homogêneos, cujo perfil de execução é dividido igualitariamente entre todos os recursos utilizados. Para simular comportamentos diferentes, são definidos múltiplos perfis de execução no simulador.

O ambiente para avaliação considera que o tamanho da fila é infinito e nenhum *job* pode ser rejeitado. Os espaços de *jobs* são limitados para os 10 primeiros seguindo a sua ordem de chegada na fila. Esta configuração demonstrou ser suficiente para permitir a análise sobre o comportamento do algoritmo no problema em questão. Ambos os ambientes de treinamento e avaliação são definidos com base na seção 3 enquanto o Batsim é utilizado somente na avaliação.

A carga de trabalho utilizada é sintética e composta por 80% de *jobs* curtos. Os tempos de execução dos *jobs* curtos são gerados seguindo uma distribuição uniforme que varia entre $[1, 3]$ unidades de tempo enquanto demais *jobs* variam entre $[10, 15]$. A quantidade de recursos de cada *job* é determinada seguindo uma distribuição uniforme dentro do intervalo definido em dois grupos, $[1, 5]$ e $[10, 15]$, escolhidos aleatoriamente. O intervalo de chegada entre os *jobs* é definido seguindo um processo de Bernoulli de modo que a carga nos recursos varie entre 10% e 190% referente a sua capacidade.

Para comparar o desempenho do algoritmo, são utilizadas quatro heurísticas tradicionais: o *Shortest Job First* (SJF), que prioriza os *jobs* com menor tempo de execução; o Packer, que prioriza os *jobs* em função da quantidade de recursos requisitada e a quantidade disponível; uma versão do Tetris, que realiza uma combinação entre o Packer e a preferência por *jobs* com o menor tempo de execução restante; e o *Easy Backfilling* (EASY), que se comporta como o *First Come First Served* (FCFS) e utiliza um mecanismo de *backfilling* para maximizar a utilização dos recursos. Além do slowdown, três outras métricas de desempenho são analisadas: o makespan, tempo para a conclusão de todos os *jobs* na carga de trabalho; o turnaround médio, tempo entre a submissão e a conclusão de cada *job*; e o tempo médio de espera na fila.

Em relação a configuração do algoritmo de treinamento, tanto a política como a função valor são representadas por uma *Feedforward Neural Network* (FNN) com apenas uma camada oculta contendo 20 unidades com função de ativação ReLU. Os parâmetros entre as redes não são compartilhados e um coeficiente de entropia de 0,01 é adicionado a função objetivo para incentivar a exploração. A taxa de aprendizado (α) é mantida constante em 0,001 e o parâmetro de corte (ϵ) é definido como 0,1. A quantidade de *actors* é definida em 24 e 64 interações são realizadas antes de ocorrer uma atualização na política. Em cada atualização, o algoritmo é otimizado por 4 épocas utilizando um total de 8*24 *minibatches*. Como *advantage function* é utilizado o *Generalized Advantage*

Estimation (GAE) com parâmetros definidos em 1 (γ) e 0,95 (λ). Esta configuração foi determinada através de uma busca informal e apresentou os melhores resultados durante a realização dos experimentos.

4.2. Resultados e Discussão

Na Figura 2, é demonstrado o desempenho das heurísticas em comparação a política de escalonamento aprendida pelo agente, usando PPO, em todas as cargas de trabalho avaliadas. É possível perceber que o algoritmo proposto se comporta igual ou melhor em todos os cenários. Houve uma redução na média do slowdown equivalente a 45%, com uma carga de 100% nos recursos, e de 54%, com uma carga de 190%, em comparação ao SJF.

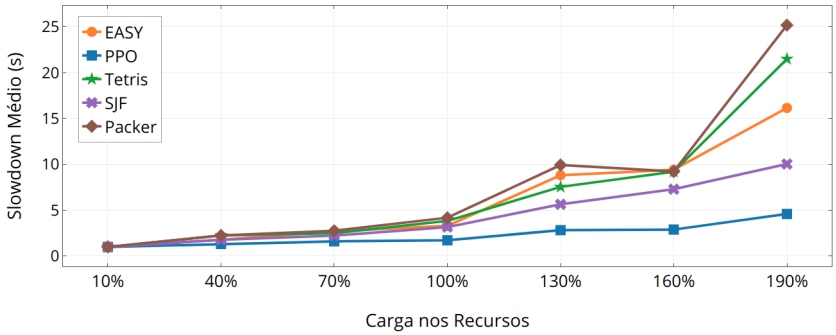

Figura 2. Comparação entre os algoritmos sob diferentes cargas de trabalho

Demais heurísticas possuem um comportamento esperado. O SJF foi melhor que as outras heurísticas por priorizar os *jobs* menores, o que reduz o slowdown. O Tetris foi melhor que o Packer por também levar em consideração o tempo dos *jobs* durante a tomada de decisão. Já o EASY demonstrou desempenho comparável ao Tetris em alguns cenários. Contudo, o seu desempenho é influenciado pela variação na ordem de chegada dos *jobs*.

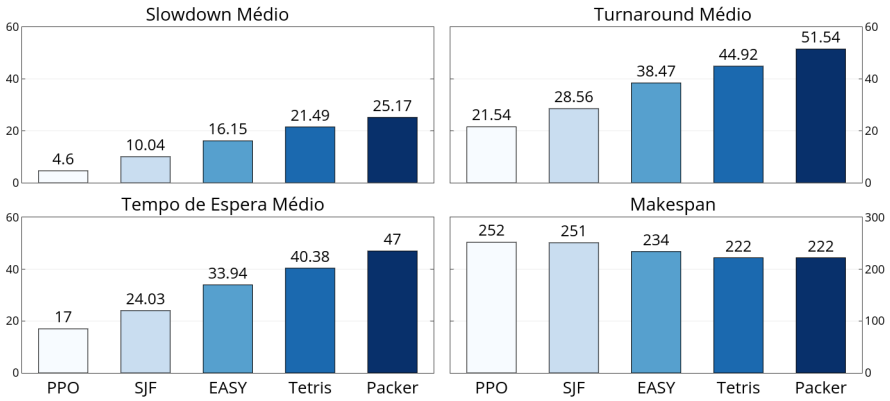

Figura 3. Desempenho dos algoritmos com carga de 190% nos recursos

Com o intuito de visualizar o impacto das escolhas do algoritmo proposto, na Figura 3 é demonstrado uma análise mais profunda considerando uma carga nos recursos de 190%. Com base nestes resultados, fica claro o trade-off entre o slowdown e o makespan. Com exceção do Tetris e do Packer, o makespan tem aumentado na medida com que a

média do slowdown diminui. Tal comportamento indica que alguns *jobs* foram atrasados por um tempo considerável, acarretando no aumento do makespan. Em comparação ao SJF, o algoritmo proposto teve uma diferença desprezível. Contudo, houve um aumento equivalente a 13,5% quando comparado ao Tetris e ao Packer. Demais métricas apresentam comportamento similar ao slowdown e indicam que, na média, o algoritmo proposto atingiu um nível maior de justiça enquanto sacrifica razoavelmente o makespan.

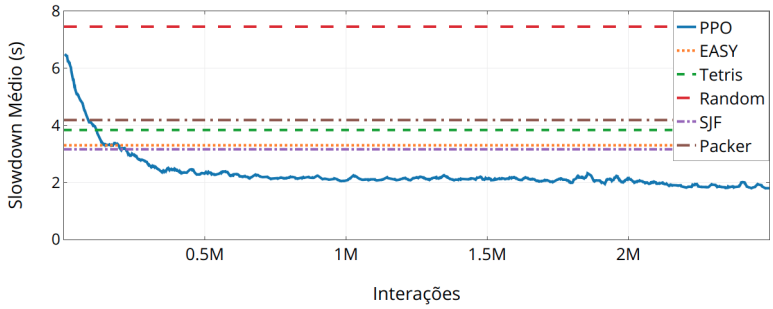

Figura 4. Curva de aprendizado

Na Figura 4 é demonstrada a curva de aprendizado do algoritmo quando treinado com a carga de 100% nos recursos. No início da interação com o ambiente, o algoritmo tem um comportamento semelhante à de uma política aleatória. Contudo, na medida com que o tempo avança e mais interações são realizadas com o ambiente, o algoritmo começa a otimizar a sua política ultrapassando o desempenho de todas as heurísticas nas primeiras 250.000 interações.

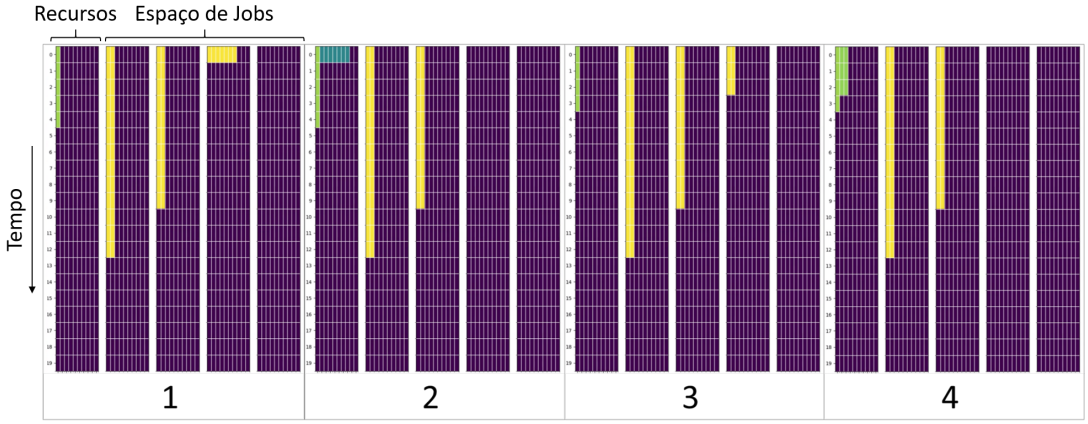

Figura 5. Estados do ambiente durante interação com o agente

Para observar o comportamento da política aprendida pelo agente, quatro quadros sequenciais dos estados do ambiente, durante interação com o agente, são coletados e apresentados de maneira parcial, contendo apenas 4 espaços de *jobs*, na Figura 5. Durante os dois primeiros quadros, é possível perceber que o agente favorece *jobs* curtos e evita alocar todos os recursos ao não escalonar os demais *jobs* na fila. No quadro 3, o agente recebe um novo *job* e o mesmo comportamento é observado no quadro 4. Logo, é possível verificar que o agente aprendeu a deixar recursos livres para os *jobs* curtos utilizarem, evitando que o sistema fique totalmente ocupado por um longo período de tempo.

Forçar deliberadamente a espera de *jobs* longos, com a finalidade de manter recursos disponíveis para os *jobs* menores, teve um impacto positivo no desempenho do algoritmo em relação ao slowdown médio. Este comportamento o distingue do SJF, que ingenuamente prioriza os menores *jobs* da fila, e apresenta maiores ganhos na medida com que a carga no sistema aumenta, pois haverá uma frequência maior de recursos disponíveis para os *jobs* menores. Em contrapartida, o atraso prolongado de alguns *jobs* pode também introduzir o efeito oposto, diminuindo a vazão do sistema sem necessariamente reduzir o slowdown.

5. Conclusão

Durante a condução deste trabalho, um algoritmo de escalonamento para computação em grids baseado em DRL foi treinado com o objetivo de minimizar o slowdown. Através da interação com um ambiente simulado, o algoritmo foi capaz de aprender uma política de escalonamento que apresentou um melhor equilíbrio entre o momento de enviar um *job* para execução e o de forçar a sua espera para que *jobs* menores tenham a preferência.

Tal comportamento obteve um ganho significativo na redução do slowdown em comparação a heurísticas tradicionais, como o SJF e o EASY. Logo, é possível concluir que métodos baseados em DRL demonstram ser uma opção viável e interessante para compor algoritmos de escalonamento adaptativo, que se adaptam aos padrões de cargas de trabalho submetidos no sistema.

Como trabalhos futuros, pretende-se analisar o algoritmo em ambientes maiores e mais representativos de uma infraestrutura em grids em produção. Não obstante, a definição do conjunto de estados e da formulação da recompensa são dois problemas que podem ser melhor explorados. Como alternativa, outras métricas de desempenho também podem ser consideradas como recompensa.

Agradecimentos

O presente trabalho foi realizado no LabP2D com apoio da UDESC, FAPESC e da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Código de Financiamento 001

Referências

Cheng, M., Li, J., and Nazarian, S. (2018). Drl-cloud: deep reinforcement learning-based resource provisioning and task scheduling for cloud service providers. In *Proceedings of the 23rd Asia and South Pacific Design Automation Conference*, pages 129–134. IEEE Press.

Foster, I., Zhao, Y., Raicu, I., and Lu, S. (2008). Cloud computing and grid computing 360-degree compared. In *Grid Computing Environments Workshop, 2008. GCE'08*, pages 1–10. Ieee.

Lelong, J., Reis, V., and Trystram, D. (2018). Tuning easy-backfilling queues. In Klusáček, D., Cirne, W., and Desai, N., editors, *Job Scheduling Strategies for Parallel Processing*, pages 43–61, Cham. Springer International Publishing.

Li, H. and Buyya, R. (2009). Model-based simulation and performance evaluation of grid scheduling strategies. *Future Generation Computer Systems*, 25(4):460–465.

Liu, N., Li, Z., Xu, J., Xu, Z., Lin, S., Qiu, Q., Tang, J., and Wang, Y. (2017). A hierarchical framework of cloud resource allocation and power management using deep reinforcement learning. In *Distributed Computing Systems (ICDCS), 2017 IEEE 37th International Conference on*, pages 372–382. IEEE.

Mao, H., Alizadeh, M., Menache, I., and Kandula, S. (2016). Resource management with deep reinforcement learning. In *Proceedings of the 15th ACM Workshop on Hot Topics in Networks*, pages 50–56. ACM.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. (2015). Human-level control through deep reinforcement learning. *Nature*, 518(7540):529.

Moghadam, M. H. and Babamir, S. M. (2018). Makespan reduction for dynamic workloads in cluster-based data grids using reinforcement-learning based scheduling. *Journal of computational science*, 24:402–412.

N'takpé, T. and Suter, F. (2018). Don't hurry be happy: A deadline-based backfilling approach. In Klusáček, D., Cirne, W., and Desai, N., editors, *Job Scheduling Strategies for Parallel Processing*, pages 62–82, Cham. Springer International Publishing.

Orhean, A. I., Pop, F., and Raicu, I. (2018). New scheduling approach using reinforcement learning for heterogeneous distributed systems. *Journal of Parallel and Distributed Computing*, 117:292 – 302.

Pooranian, Z., Shojafar, M., Abawajy, J. H., and Abraham, A. (2015). An efficient meta-heuristic algorithm for grid computing. *Journal of Combinatorial Optimization*, 30(3):413–434.

Poquet, M. (2017). *Simulation approach for resource management*. Theses, Université Grenoble Alpes.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017). Proximal policy optimization algorithms. *CoRR*, abs/1707.06347.

Sutton, R. S., Barto, A. G., Bach, F., et al. (1998). *Reinforcement learning: An introduction*. MIT press.

Tong, Z., Xiao, Z., Li, K., and Li, K. (2014). Proactive scheduling in distributed computing - a reinforcement learning approach. *Journal of Parallel and Distributed Computing*, 74(7):2662–2672.

Vasile, M.-A., Florin, P., Mihaela-Cătălina, N., and Cristea, V. (2018). Mlbox: Machine learning box for asymptotic scheduling. *Information Sciences*, 433:401–416.

Wu, J. and Xu, X. (2018). Decentralised grid scheduling approach based on multi-agent reinforcement learning and gossip mechanism. *CAAI Transactions on Intelligence Technology*, 3(1):8–17.