Uma Comparação entre Escalonadores de Tarefas baseados em Aprendizado de Máquina

Claudinei Cabral Junior[1], **Guilherme Piêgas Koslovski**[1]

[1]Departamento de Ciência da Computação
Universidade do Estado de Santa Catarina (UDESC) – Joinville, SC – Brasil

claudinei.cj@edu.udesc.br, guilherme.koslovski@udesc.br

Resumo. *O escalonamento de tarefas desempenha um papel fundamental em sistemas computacionais, impactando diretamente na eficiência e no uso de recursos. Algoritmos tradicionais enfrentam dificuldades em ambientes dinâmicos, tornando abordagens baseadas em aprendizado de máquina uma alternativa promissora. Dentre as abordagens existentes, este trabalho compara os escalonadores Decima e ACRL, baseados em Reinforcement Learning.*

1. Introdução

O escalonamento de tarefas e a seleção de quais servidores devem executá-las desempenham papéis importantes em sistemas computacionais. Escalonadores eficientes são cruciais, visto que impactam diretamente na utilização eficiente dos recursos computacionais, na redução de custos administrativos e no desempenho das aplicações. Conforme ambientes computacionais se tornam mais complexos e dinâmicos, algoritmos de escalonamento tradicionais revelam-se ineficazes. Nesse contexto, técnicas de aprendizado de máquina têm emergido oferecendo uma melhor abordagem para geração e otimização de políticas de escalonamento devido à tomada de decisões dinâmicas e adaptáveis aos contextos específicos.

Dentre as abordagens baseadas em aprendizado de máquina, destacam-se nesse artigo duas metodologias: *Decima* [Mao et al. 2019], que utiliza *Reinforcment Learning* (RL) juntamente com redes neurais para treinar melhores políticas de escalonamento diante de uma carga de trabalho, e o *Actor-Critic Reinforcement Learning* (ACRL), que possui uma arquitetura distinta, atribuindo papéis à seleção das ações (*Actor*) e avaliação das ações (*Critic*) [Koslovski et al. 2024]. Ambas abordagens visam superar as limitações de políticas de escalonamento tradicionais, como a capacidade de lidar com dependências complexas entre tarefas e aleatoridade na chegada de trabalhos [Feitelson and Rudolph 1998].

O presente trabalho apresenta uma proposta de análise comparativa entre os escalonadores de tarefas supracitados, demonstrando alguns resultados preliminares. Inicialmente, os modelos são treinados com a mesma base de conhecimento, sendo posteriormente comparados usando dados novos. Para realizar a comparação, foram selecionadas as métricas tempo atual de execução, uso de servidores, número de tarefas e trabalhos escalonados, além do tempo de tarefas em espera.

2. Escalonadores baseados em *Reinforcement Learning*

A proposta Decima [Mao et al. 2019] utiliza técnicas de *Reinforcement Learning* (RL) e redes neurais para aprender políticas de escalonamento específicas para a carga de trabalho recebida, visando alcançar um objetivo de alto nível, como minimizar o tempo médio

de conclusão de trabalhos. Decima adapta e otimiza suas decisões de escalonamento levando em conta dependências entre estágios, codificadas em grafos acíclicos direcionados, e monitorando informações e registros de cargas de trabalho passadas. Com base nestas informações, o modelo evita políticas genéricas e aloca recursos dinamicamente com base nas necessidades de cada carga de trabalho. Com o suporte de sua rede neural, o treinamento do modelo ocorre através de uma série de experimentos. Em cada iteração, o modelo realiza o escalonamento uma carga de trabalho, examina o resultado obtido e, com base no *feedback*, gradativamente melhora sua política de escalonamento.

Por sua vez, a política de escalonamento *Actor-Critic* (AC) *Reinforcement Learning* (RL) consiste em uma arquitetura distinta, dividindo o processo de decisão em dois componentes fundamentais que diferem de algoritmos de escalonamentos tradicionais. O *Actor* é responsável por tomar uma decisão selecionando ações com base na política atual do ambiente. Sua função é explorar o espaço de ação a fim de escolher aquelas que maximizam a recompensa acumulada esperada. Com base em *feedback* fornecido pelo *Critic* e pelo ambiente, o *Actor* tenta melhorar sua política, se adaptando ao ambiente dinâmico que está inserido. O *Critic* tem o papel de avaliar as ações tomadas pelo *Actor*. Para isso, ele calcula um valor estimado para cada decisão, com base no desempenho observado e no *feedback* do ambiente. Se a decisão realizada pelo *Actor* resultou em um bom desempenho, o *Critic* irá reforçar essa escolha, fornecendo uma guia ao *Actor* para que realize ações semelhantes no futuro. Em suma, através de um processo iterativo, o *Actor* e *Critic* trabalham em conjunto para aprimorar sua política de escalonamento, melhorando sua capacidade de lidar com diferentes cargas de trabalho.

3. Desenvolvimento

Inicialmente, os modelos foram configurados para atender aos mesmos requisitos, garantindo que as condições de treinamento fossem equivalentes. Para isso, foram estabelecidos parâmetros comuns, como o número de episódios e servidores, de modo que os escalonadores pudessem ser treinados sob condições comparáveis. A fim de assegurar uma avaliação justa, foi necessário obter uma carga de trabalho que permitisse que ambos os modelos usufruíssem. O modelo Decima destacou-se por possuir uma base de dados bem distribuída, garantindo aleatoriedade na chegada de trabalhos e tarefas. Dados relevantes, como dependências entre estágios, número de tarefas e suas respectivas durações foram extraídos para análise. No entanto, devido à arquitetura específica do Decima, foi necessário gerar artificialmente dependências entre tarefas (DAGs), a partir das dependências entre estágios. Cada estágio foi decomposto, extraindo todas tarefas presentes e criando dependências entre estas tarefas. Para preservar a dependência entre os estágios, foi criada uma adjacência entre a última tarefa de cada estágio pai com a primeira tarefa do estágio filho. É importante ressaltar que tais alterações não afetam a implementação original do Decima, alterando apenas os dados de entrada, para alinhar o cenário de simulação.

Com a carga de trabalho definida, os modelos foram submetidos ao processo de treinamento. Para tal etapa, Decima e ACRL receberam exatamente a mesma base de conhecimento e foram treinados com os parâmetros definidos em suas publicações originais. Posteriormente, iniciou-se a fase de escalonamento usando uma nova base de dados, idêntica para os dois modelos. Durante a simulação dos escalonamentos, as métricas de desempenho foram monitoradas e armazenadas a fim de realizar uma comparação pos-

terior. Especificamente, as métricas foram armazenadas a cada evolução temporal dos simuladores. Por fim, devido às diferenças na representação das escalas temporais entre os modelos, foi necessário normalizar as métricas relacionadas ao tempo total. A normalização foi feita dividindo cada valor pelo valor máximo da métrica, o que resultou em valores variando de 0 a 1.

4. Simulações e resultados

Para realizar as simulações, os parâmetros descritos na Tabela 1 foram definidos. Embora o número de trabalhos seja relativamente baixo, cada um contém um grande volume de tarefas, resultando em uma alta carga de processamento. Para lidar com essa demanda, foram alocados 50 servidores. Além disso, para assegurar um treinamento eficaz dos modelos, foram executados 100 episódios, com dois agentes treinando o modelo de forma independente.

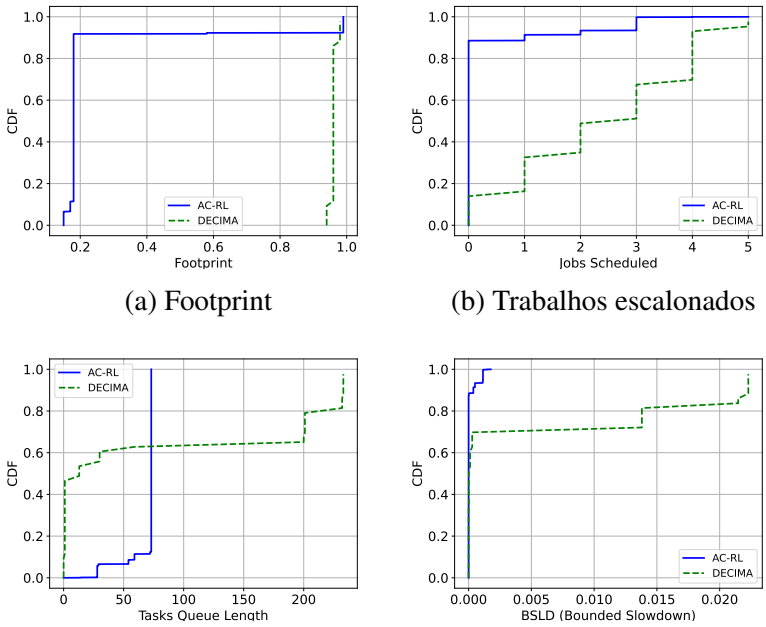
Parâmetro	Descrição	Valor
Servidores	Recursos computacionais responsáveis por processar e executar tarefas	50
Episódios	Sequência completa de iterações entre o agente e o ambiente	100
Número de trabalhos	Unidades de processamento que podem possuir características específicas	5
Número de tarefas	Unidades de operações a serem executadas	560
Número de agentes	Instâncias independentes que tomam decisões no ambiente	2

Tabela 1: Definição de parâmetros para execução do protocolo de simulação.

Durante o processo de escalonamento, as métricas foram extraídas para ambos os modelos, incluindo: tempo atual de execução, *footprint* (percentual de servidores em uso), número de trabalhos e tarefas em fila, número de trabalhos e tarefas escalonados, *slowdown* (tempo acumulado das tarefas na fila) e *bounded slowdown*.

A Figura 1 apresenta as métricas usando *Cumulative Distribution Functions (CDFs)*: (a) *Footprint*, (b) Trabalhos escalonados, (c) Tarefas escalonadas e (d) *Bounded Slowdown*. No eixo X, estão representadas as métricas, enquanto o eixo Y exibe a CDF que indica a porcentagem de valores que são menores ou iguais a um ponto X. A Figura 1(a) revela que o modelo *ACRL* manteve um baixo consumo de recursos, abaixo de 20%, enquanto *Decima* teve uma distribuição mais concentrada consumindo mais de 85% dos recursos dos servidores. Já a Figura 1(b) indica que o modelo *Decima* desempenhou melhor, com uma distribuição uniforme, escalonando trabalhos mais eficiente. Ainda sim, o modelo *ACRL* foi capaz de escalonar os trabalhos rapidamente demonstrando uma performance satisfatória. A Figura 1(c) apresenta o *ACRL* com uma distribuição satisfatória de tarefas, com uma fila menor, diminuindo o tempo de espera para tarefas serem executadas. Por fim, a Figura 1(d) evidencia menores valores de *bsld* para o modelo *ACRL*, sugerindo que o escalonador minimiza melhor os atrasos.

Os resultados preliminares indicam que mesmo utilizando modelos teoricamente adaptáveis à diferentes contextos, é necessário realizar uma análise considerando múltiplas bases de conhecimento. Especificamente, a continuação do trabalho buscará comparar os escalonadores em diferentes cenários relacionados com a ocupação total da infraestrutura.



(a) Footprint (b) Trabalhos escalonados

(c) Tarefas em espera (d) Bounded Slowdown

Figura 1: Comparação entre métricas

5. Conclusão

Este trabalho contribuiu para o entendimento de escalonadores de tarefas baseados em aprendizado por reforço, fornecendo uma análise comparativa entre os modelos *Decima* e *ACRL*. Os resultados preliminares indicaram que apesar dos modelos apresentarem bom desempenho em algumas métricas, é necessário uma análise mais ampla, abrangendo múltiplas bases de conhecimento. A continuidade do estudo focará na comparação dos escalonadores buscando compreender melhor seu desempenho em cenários variados de ocupação total da infraestrutura.

Agradecimentos: Este trabalho recebeu apoio do Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq), Fundação de Amparo à pesquisa e Inovação (FAPESC), desenvolvido no Laboratório de Processamento Paralelo e Distribuído (LabP2D).

Referências

Feitelson, D. G. and Rudolph, L. (1998). Metrics and benchmarking for parallel job scheduling. In Feitelson, D. G. and Rudolph, L., editors, *Job Scheduling Strategies for Parallel Processing*, pages 1–24, Berlin, Heidelberg. Springer Berlin Heidelberg.

Koslovski, G. P., Pereira, K., and Albuquerque, P. R. (2024). Dag-based workflows scheduling using actor–critic deep reinforcement learning. *Future Generation Computer Systems*, 150:354–363.

Mao, H., Schwarzkopf, M., Venkatakrishnan, S. B., Meng, Z., and Alizadeh, M. (2019). Learning scheduling algorithms for data processing clusters. In *Proceedings of the ACM Special Interest Group on Data Communication*, SIGCOMM '19, page 270–288, New York, NY, USA. Association for Computing Machinery.