Usando *Deep Learning* para o Escalonamento de Tarefas Comunicantes

Kleiton Pereira[1], **Guilherme Piêgas Koslovski**[1]

[1]LabP2D - Universidade do Estado de Santa Catarina – UDESC

Resumo. *Este trabalho apresenta a proposta de um agente escalonador de tarefas comunicantes baseado em Deep Learning. O agente é treinado utilizando algoritmos conhecidos da literatura, amalgamando suas decisões em busca de generalizar o problema de escalonamento a partir da combinação dos processos de algoritmos simples e eficientes. Os resultados indicam que o agente escalonador obteve resultados competitivos comparado aos algoritmos tradicionais.*

1. Introdução

O problema do escalonamento pode ser definido como o ato de dividir e sequenciar elementos a fim de encontrar uma permutação que otimize um ou mais parâmetros em um determinado período de tempo. Embora o problema não seja novo, o surgimento de sistemas computacionais modernos elevou a complexidade para um novo patamar, dado o número elevado de elementos que permeiam tais sistemas, o que implica em maior dificuldade em realizar o escalonamento ótimo em tempo hábil. É possível dividir todo o processo de escalonamento em 3 partes: o pré-escalonamento, a tomada de decisão do escalonamento e o pós-escalonamento. O presente trabalho aborda apenas as duas primeiras etapas. O pré-escalonamento abrange todos os processos anteriores a tomada de decisão (*e.g.*, formação de uma fila de tarefas), já a tomada de decisão é o escalonamento sendo realizado, decidindo onde e quando uma tarefa terá início.

As diversas abordagens para o escalonamento variam de acordo com o sistema, os recursos e os objetivos que devem ser otimizados. Neste trabalho definimos tarefas comunicantes como qualquer conjunto de tarefas em que ocorre troca de mensagens (comunicação) interna. Estas tarefas podem ser modeladas através de um *Directed Acyclic Graph* (DAG) $G(V, E)$, no qual V são os vértices do grafo, sendo $V = T_1, T_2, T_3, ..., T_N$ representando as N tarefas que demandam recursos de processamento, e E são as arestas. Uma aresta $E_{ij} = (T_i, T_j)$ denota uma troca de mensagens que parte de T_i e termina em T_j. Pesos podem ser adicionados ao grafo para denotar quão custosa será a comunicação.

A aplicação de Inteligência Artificial (IA) ao problema de escalonamento não é algo novo, com trabalhos apresentando o conceito datando dos anos 80 [Aytug et al. 1994]. Porém, com o advento de recursos computacionais sob demanda, unido aos avanços das técnicas de IA, principalmente no campo de aprendizado de máquina [Zhang et al. 2018], há interesse na aplicação das novas técnicas ao problema, principalmente dada a natureza NP-Difícil do problema de escalonamento, inviabilizando soluções ótimas determinísticas para escalas atuais, por este motivo buscam-se soluções heurísticas que se aproximem da solução ótima em tempo hábil.

2. Trabalhos Relacionados

Na Tabela 1 são elencados os trabalhos relacionados, apresentando qual método de Aprendizado de Máquina foi utilizado para o treinamento, se há escalonamento de tarefas sem

dependência ou com dependência, bem como se a rede e a comunicação são levadas em consideração pelas soluções propostas e, por fim, em qual etapa atua a solução. As principais tendências observadas na literatura são a fraca presença da rede na tomada de decisão ou no processo de treinamento do algoritmo, além de uma divisão entre Aprendizado Supervisionado e Aprendizado por Reforço. A maioria dos trabalhos foca nas etapas de Pré-escalonamento e Escalonamento, o que é esperado, dado o impacto que essas etapas possuem no processo de escalonamento.

Trabalho	Método	S/ Dep.	C/ Dep.	Rede	Etapa
[Vasile et al. 2018]	Supervisionado	✓	✓	✗	Pré
[Blenk et al. 2017]	Supervisionado	-	-	-	Não se aplica
[Mao et al. 2016]	Reforço	✓	✗	✗	Escalonamento
[Cheng et al. 2018]	Reforço	✗	✓	✓	Pré e Escalonamento
[Woo et al. 2018]	Reforço	✗	✓	✗	Escalonamento
[Weckman et al. 2008]	Supervisionado	✗	✓	✗	Pré
[Tripathy et al. 2015]	Supervisionado	✗	✓	✓	Escalonamento

Tabela 1. Trabalhos Relacionados.

3. Proposta

A solução proposta, denominada KAIROS (*k-Graphs Artificial Intelligence-based Request Optimization Scheduler*) consiste em um escalonador de tarefas comunicantes baseado em técnicas recentes de IA e *Deep Learning*.

Ambiente. O ambiente é representado por uma simulação na qual requisições são tratadas e escalonadas pelo agente e entregues para alocação. O ambiente é encarregado de ajustar os valores internos, simulando o consumo dos recursos, bem como a passagem de tempo utilizando eventos discretos. Além disso, o ambiente fornece ao agente um *snapshot* do estado atual do ambiente, contendo dados sobre o uso de recursos e eventos passados, bem como requisições que já estão na fila de tarefas, fornecendo uma noção temporal sequencial para que haja um escalonamento levando em consideração múltiplas tarefas e suas possíveis comunicações. O propósito do ambiente é ser simples e genérico, podendo representar, através de sua configuração inicial, diferentes arquiteturas de sistemas modernos.

Agente. O agente atua nas etapas de pré-escalonamento e tomada de decisão, escalonando um ou mais conjuntos de tarefas representados por DAGs. Inicialmente, as tarefas são organizadas em uma fila que alimenta o algoritmo, e este por sua vez gera uma saída que representa a tomada de decisão do escalonamento. O agente escolhe o melhor recurso de processamento e a melhor unidade de tempo para o início de cada tarefa, levando em consideração a alocação das outras tarefas da fila, com intenção de otimizar uma ou mais métricas pré-selecionadas. As técnicas de *Deep Learning* e Redes Neurais são aplicadas para alcançar este objetivo. Entretanto, há muitos parâmetros que podem variar, como número de camadas, quantidade de nós por camada, arquitetura da rede, funções de ativação, entre outros.

Na fase de treinamento, são utilizados diversos algoritmos tutores executados em paralelo (*e.g.,* Round Robin, *First-Come First Served*) e, a cada iteração, verifica-se o desempenho de acordo com as métricas que deseja-se otimizar. O processo de treinamento é iniciado baseado nos algoritmos tutores com melhor resultado. Os pesos são ajustados e o processo repete-se para cenários diferentes. Após diversas iterações espera-se que

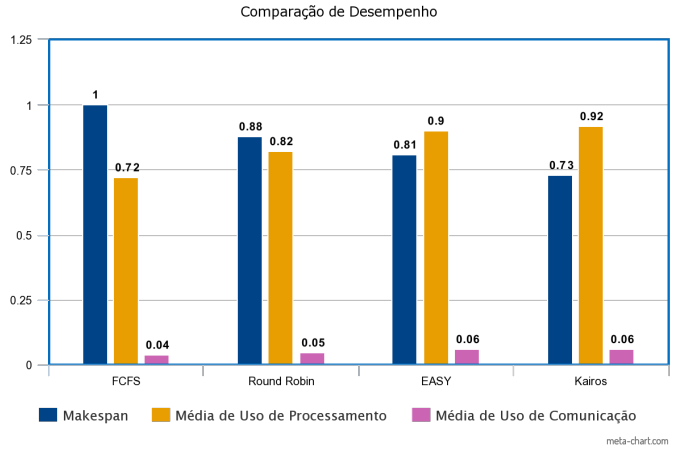

Figura 1. Desempenho comparativo preliminar.

o agente consiga abstrair o processo de escalonamento para cada métrica selecionada, amalgamando os processos de tomada de decisão dos algoritmos tutores, ultrapassando os mesmos, o que é definitivamente possível [Tripathy et al. 2015].

Ferramentas para Implementação. Foram utilizada as linguagens Python e C, com as bibliotecas Tensorflow, OpenAI Gym e Numba.

4. Análise Experimental Preliminar

Na Figura 1 é possível ver um desempenho comparativo do algoritmo em sua fase inicial de produção. Os dados foram coletados com a simulação baseada em carga semi-sintética. O algoritmo foi treinado pelos algoritmos tutores FCFS, Round Robin e EASY utilizando as seguintes métricas de seleção: *Makespan*, que é o tempo total necessário para execução das tarefas; Média de Uso de Processamento, sendo a média de tempo em que os recurso de processamento estavam ativos; e Média de Uso de Comunicação, que é similar a métrica anterior, porém levando em consideração os recursos de comunicação.

Optou-se por utilizar dados semi-sintéticos, em que tem-se a geração de uma carga sintética que é baseada em uma carga real. Foram geradas cargas tendo como base diferentes problemas do NPB (*NAS Parallel Benchmark*). Tais problemas foram analisados em termos de processamento e comunicação e, a partir disto, múltiplas distribuições normais foram geradas. Tendo posse destas distribuições é possível gerar cargas sintéticas baseadas em cargas reais. Sendo os problemas do NPB bastante divergentes quanto seus custos de processamento e comunicação isso torna a carga bastante heterogênea, o que serve para analisar a sensibilidade do algoritmo a diferentes conjuntos de requisições em análises mais complexas. A heterogeneidade da carga também serve como dificuldade a ser superada pelo algoritmo, obrigando-o a generalizar o processo de escalonamento, diminuindo as chances do algoritmo decorar soluções. Para representar o *data center* foi utilizada a topologia *Fat Tree* com $k = 4$, totalizando 16 pontos de processamento homogêneos com 8 unidades de processamento cada, e 10 pontos de comunicação, dentre estes 8 possuíam 50 unidades de comunicação e 2 possuíam 100 unidades de comunicação. Foram geradas 300 requisições semi-sintéticas escolhidas entre as diferentes classes de problemas do NPB para serem escalonadas em uma janela de 500 unidades de tempo.

Os dados da Figura 1 foram normalizados para melhor visualização. Nas colunas de *Makespan* valores menores indicam um tempo menor para a conclusão de todas as tarefas, já nas outras colunas, de Média de Uso de Processamento e Comunicação, valores maiores indicam um menor tempo ocioso dos recursos. Os resultados, embora ainda não representem o potencial final do algoritmo, já indicam um ganho em relação aos algoritmos tutores, possuindo desempenho igual ou melhor nas três métricas coletadas, indicando o importante fato de que com técnicas de IA e *Deep Learning*, o agente pode atingir um desempenho melhor do que os algoritmos utilizados para o seu aprendizado.

5. Conclusões

Neste trabalho foram apresentados o problema de escalonamento e uma proposta para solução usando *Deep Learning* para a automatização de diversas etapas para o escalonamento. Os resultados experimentais indicam que *Deep Learning* obteve resultados competitivos quando comparado com algoritmos tradicionais.

Agradecimentos: LabP2D, UDESC e FAPESC.

Referências

Aytug, H., Bhattacharyya, S., Koehler, G. J., and Snowdon, J. L. (1994). A review of machine learning in scheduling. *IEEE Transactions on Engineering Management*, 41(2):165–171.

Blenk, A., Kalmbach, P., Kellerer, W., and Schmid, S. (2017). o'zapft is: Tap your network algorithm's big data! In *Proceedings of the Workshop on Big Data Analytics and Machine Learning for Data Communication Networks*, pages 19–24. ACM.

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.

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.

Tripathy, B., Dash, S., and Padhy, S. K. (2015). Dynamic task scheduling using a directed neural network. *Journal of Parallel and Distributed Computing*, 75:101–106.

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

Weckman, G. R., Ganduri, C. V., and Koonce, D. A. (2008). A neural network job-shop scheduler. *Journal of Intelligent Manufacturing*, 19(2):191–201.

Woo, S., Yeon, J., Ji, M., Moon, I.-C., and Park, J. (2018). Deep reinforcement learning with fully convolutional neural network to solve an earthwork scheduling problem. In *2018 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pages 4236–4242. IEEE.

Zhang, Q., Yang, L. T., Chen, Z., and Li, P. (2018). A survey on deep learning for big data. *Information Fusion*, 42:146–157.