Usando aprendizado supervisionado para composição de políticas de escalonamento

Guilherme Diel[1], **Ana Eloina Nascimento Kraus**[1], **Guilherme Piêgas Koslovski**[1]

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

Resumo. *Este artigo propõe a utilização de aprendizado supervisionado para melhorar as políticas de escalonamento em ambientes de computação de alto desempenho, utilizando dados que incorporam políticas existentes (algoritmos tutores). O modelo treinado revela-se adaptável, superando, em alguns casos, as limitações dos algoritmos tutores.*

1. Introdução

A Computação de Alto Desempenho (HPC, de *High Performance Computing*) envolve a coordenação de diversos computadores para cálculos paralelos. O escalonamento, em HPC, refere-se à definição da ordem de execução das tarefas submetidas pelos usuários, sendo complementado pela alocação eficiente de recursos como servidores, processadores, memória e rede de comunicação. Para otimizar ambientes HPC, é crucial aprimorar as políticas de escalonamento para mitigar custos operacionais e aumentar a produtividade [Carastan-Santos et al. 2019].

Este artigo utiliza técnicas de aprendizado supervisionado, analisando dados de escalonamentos anteriores realizados por algoritmos oriundos da literatura especializada, denominados tutores. Em geral, o estudo realiza a análise considerando uma base de dados populada com três informações principais: *(i)* as requisições submetidas pelos usuários; *(ii)* a descrição das infraestruturas de HPC; e *(iii)* as métricas obtidas após a execução de cada política de escalonamento. Tais informações são utilizadas para treinar modelos, que posteriormente são utilizados para composição de políticas de escalonamento. De forma geral, o objetivo principal é apresentar resultados da aplicação de aprendizado supervisionado para melhorar políticas de escalonamento em sistemas HPC. Assim, são detalhados a composição da base de dados, o protótipo implementado e os resultados das simulações. O artigo está estruturado com a Seção 2 detalhando a base de dados, a Seção 3 apresentando resultados, e as considerações finais na Seção 4.

2. Desenvolvimento do escalonador

Este trabalho emprega o método de aprendizado supervisionado, no qual um modelo é treinado utilizando conjuntos de dados rotulados. Cada conjunto é constituído por uma matriz de números representando os dados (X) e um rótulo associado (Y). O modelo analisa esses conjuntos em busca de padrões entre X e Y. No âmbito desta pesquisa, X representa a métrica alvo da otimização, enquanto Y corresponde à posição geral dessa métrica em uma fila. O banco de dados utilizado é composto por informações abrangentes, incluindo todas as tarefas submetidas, o estado dos recursos computacionais e as métricas obtidas ao aplicar diferentes políticas. Em outras palavras, o banco de dados é formado por diversos algoritmos tutores. O modelo escolhido para treinar a inteligência artificial é o Naive Bayes Gaussiano (GNB), devido à sua eficiência computacional, conforme destacado em [Ontivero-Ortega et al. 2017] e corroborado por testes preliminares.

2.1. Trabalhos, tarefas e métricas

Para treinar o modelo, é essencial ter dados e seus rótulos. Os dados foram gerados a partir de dois bancos de dados: SDSC-SP2 e HPC2N, oriundos de [Feitelson et al. 2014]. Vale ressaltar que essas bases apresentam diferenças significativas entre si, como a quantidade total de tarefas, a disponibilidade de recursos em cada uma delas e o tempo de chegada individual de cada tarefa. Cada tarefa (t) é caracterizada por seis atributos: tempo de processamento solicitado (p_t), recursos computacionais solicitados (q_t) e tempo de chegada (r_t), conhecidos no momento da requisição. Após o escalonamento, o tempo de início (s_t) e o tempo de espera (w_t) são registrados. Ao término da execução, o tempo total de execução (f_t) é quantificado. O tempo de espera (w_t) de cada tarefa é calculado como $w_t = s_t - r_t$. Seguindo essa lógica, a métrica *bounded slowdown* (*bsld*) para uma dada tarefa é definido como $bsld_t = \max\left\{\frac{w_t+p_t}{\max(p_t,\tau)}, 1\right\}$, sendo τ uma constante ajustada em torno de 10 segundos, evitando que tarefas pequenas tenham valores excessivamente grandes [Feitelson and Rudolph 1998, Carastan-Santos et al. 2019]. Após o escalonamento de todos os bancos de dados utilizando as políticas de escalonamento dos algoritmos tutores e obtenção do *bsld* para todas as tarefas, a fila é organizada com base no *bsld*. Isso possibilita fornecer como entrada o *bsld* como X e a posição na fila como Y.

2.2. Algoritmos tutores

Existem diversas pesquisas dedicadas à criação de novas políticas de escalonamento, no entanto, é crucial reconhecer que cada política apresenta vantagens e desvantagens que variam de acordo com as características específicas do cenário em que as tarefas são executadas. Observa-se uma tendência crescente no uso de políticas consolidadas para compor conjuntos de dados de conhecimento, ampliando o escopo de pesquisa no aprimoramento de cenários de tomada de decisão [Carastan-Santos and de Camargo 2017, Carastan-Santos et al. 2019, Koslovski et al. 2024]. Nesse contexto, as políticas de escalonamento consideradas incluem o *First Come First Serve* (FCFS); o *Shortest Area First* (SAF); o *Shortest Requested Resources First* (SQF); e o *Shortest Processing Time* (SPT). Ainda, as funções F1 a F4 adotam uma abordagem composta [Carastan-Santos and de Camargo 2017]. Os cenários de simulação consideram dois fatores: a quantidade de servidores disponíveis e a energia consumida durante o processamento. É importante observar que cada servidor tem a capacidade de processar apenas uma tarefa por vez, e é imperativo que nenhuma tarefa seja interrompida.

2.3. Definição dos rótulos e treinamento

Na fase de definição dos rótulos, considerando que o modelo busca prever o rótulo com base em X, e com o objetivo de otimizar a métrica global do *bsld*, foi determinado que esta métrica seria designada como Y. Dessa forma, cada tarefa é associada a um valor específico de *bsld*, dependendo do algoritmo tutor utilizado durante o escalonamento. A ideia subjacente é que o modelo será treinado para selecionar o algoritmo tutor que resultou no menor *bsld* para uma determinada tarefa. Assim, a escolha do rótulo pelo modelo será orientada pela minimização do *bsld*. O processo de treinamento foi conduzido mediante a simulação de escalonamento de bases de dados, utilizando algoritmos tutores no *framework* Batsim. Vale ressaltar que, para uma mesma tarefa, a utilização de algoritmos tutores distintos para o escalonamento pode resultar em diferentes valores de *bsld*. Após a obtenção dos valores de *bsld* para cada tarefa, foi formada uma fila organizada em ordem crescente, usada como base para o treinamento do modelo.

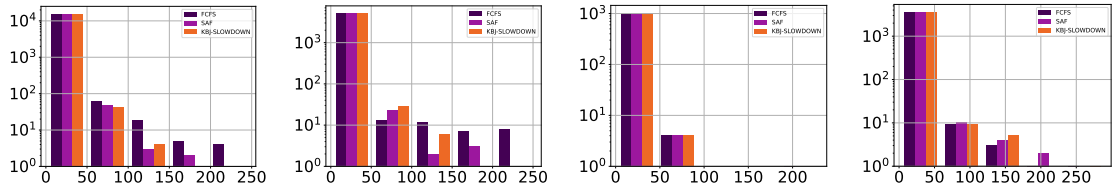
3. Simulações e resultados

Para conduzir as simulações, duas ferramentas foram utilizadas: Batsim e Pybatsim. O Batsim, um *framework* que funciona como plataforma de simulação, desempenhou o papel dos usuários, enviando trabalhos ao Pybatsim, interface Python para interagir com essa plataforma, que atuou como servidor de escalonamento. No entanto, em vez de processar efetivamente as tarefas, o Pybatsim considerou apenas o tempo estimado para a execução, avançando o tempo correspondente. No ambiente do Pybatsim, os escalonadores tutores e o modelo treinado foram implementados. O fluxo operacional foi o seguinte: ao chegar um novo trabalho, ele é adicionado à fila de espera, que é escalonada conforme alguma política de escalonamento (utilizando um dos algoritmos tutores ou o modelo treinado). Quando o primeiro processo inicia a simulação, ele é removido da fila e passa a simular a utilização dos recursos do Batsim. Ao término da simulação, os recursos são devolvidos ao Batsim, e a próxima tarefa na fila é escalonada, se possível.

3.1. Métricas e cenários

A métrica primária adotada para a avaliação dos resultados é o *bsld*, que se alinha com o objetivo de garantir uma análise consistente e direcionada para a eficácia da política de escalonamento [Feitelson and Rudolph 1998]. Para assegurar a imparcialidade dos resultados, os dados analisados foram gerados utilizando uma base de dados que não foi utilizada no treinamento do modelo (SDSC-BLUE). Quatro histogramas foram gerados, sendo que Y indica a frequência de ocorrência de dados em X. Esses histogramas são categorizados de acordo com as tarefas, segmentando os valores de p_t em intervalos específicos: *(i)* $p_t \leq 25\% p_t^{max}$; *(ii)* $25\% p_t^{max} < p_t \leq 50\% p_t^{max}$; *(iii)* $50\% p_t^{max} < p_t \leq 75\% p_t^{max}$; e *(iv)* $75\% p_t^{max} < p_t \leq 100\% p_t^{max}$. Essa abordagem de categorização permite uma análise mais detalhada da distribuição dos dados em relação aos diferentes intervalos de p_t.

3.2. Discussão



(a) 0-25%. (b) 25-50%. (c) 50-75%. (d) 75-100%.

Figura 1. Resultados (métrica *bsld*) organizados pelos valores de p_t.

A Figura 1(a)-1(d) apresenta uma comparação entre dois principais algoritmos tutores (FCFS e SAF), os quais foram identificados previamente como detentores dos resultados promissores [Carastan-Santos et al. 2019]. Paralelamente, os gráficos também demonstram o uso do modelo proposto, denominado KBJ-Slowdown, em que em X estão os valores de *bsld* e Y a quantidade de vezes em que aparecem. A Figura 1(a), 1(b) e 1(d) evidenciam que o modelo desenvolvido demonstrou desempenho tão bom quanto ou superior ao tutor mais eficiente. Isso é corroborado pelo fato de que a quantidade de resultados elevados (> 150) é menor ou igual àquela dos tutores, ao mesmo tempo em que os demais resultados são similares. Porém, é possível notar que o modelo gerou mais resultados com *bsld* entre 100 e 150 (próximos a resultados elevados) comparado

ao melhor tutor. Por outro lado, a Figura 1(c) indica que, de maneira geral, o modelo apresentou resultados iguais aos escalonadores tutores. Observa-se que o modelo exibe similaridades com o desempenho do algoritmo tutor mais eficiente.

Dessa forma, é possível concluir que o modelo assimilou os padrões dos algoritmos tutores durante o aprendizado, resultando em uma política de escalonamento que, em sua maioria, minimiza a ocorrência de valores elevados de *bsld*. Ademais, o modelo demonstra sua capacidade de superar as limitações dos algoritmos tutores, apresentando um desempenho superior em determinadas condições operacionais. Essa adaptabilidade destaca a promissora aplicação de técnicas de aprendizado de máquina na otimização de políticas de escalonamento em ambientes computacionais complexos.

4. Conclusão

Este trabalho contribuiu para a compreensão da aplicabilidade das técnicas de aprendizado supervisionado na otimização de políticas de escalonamento em sistemas HPC. A aplicação de técnicas de aprendizado supervisionado, com foco no escalonamento de bases de dados por meio de algoritmos tutores, apresentou resultados promissores, apontando para um caminho significativo na busca por soluções eficientes e adaptáveis em ambientes HPC. As análises e resultados apresentados abrem portas para futuras investigações, incluindo a exploração de outras métricas.

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). 3

Referências

Carastan-Santos, D. and de Camargo, R. Y. (2017). Obtaining dynamic scheduling policies with simulation and machine learning. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '17, New York, NY, USA. Association for Computing Machinery.

Carastan-Santos, D., De Camargo, R. Y., Trystram, D., and Zrigui, S. (2019). One can only gain by replacing easy backfilling: A simple scheduling policies case study. In *2019 19th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*, pages 1–10.

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

Feitelson, D. G., Tsafrir, D., and Krakov, D. (2014). Experience with using the parallel workloads archive. *Journal of Parallel and Distributed Computing*, 74(10):2967–2982.

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.

Ontivero-Ortega, M., Lage-Castellanos, A., Valente, G., Goebel, R., and Valdes-Sosa, M. (2017). Fast gaussian naïve bayes for searchlight classification analysis. *NeuroImage*, 163:471–479.