Modelo BUCMA: Beneficiador de Usuários Conscientes MultiAlgoritmo em grades computacionais

Geremias Corrêa, Maurício A. Pillon e Charles C. Miers[1]

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

geremias.correa@edu.udesc.br, {mauricio.pillon,charles.miers}@udesc.br[1]

Abstract. *Computational grids provide on-demand computing resources based on policy-defined allocations implemented in scheduling algorithms. The walltime of a request designates the estimated provisioning time of the tasks, but it is crudely used. We propose BUCMA, a user bonification/punishment multialgorithm model based on our walltime estimation. In the long term, the objective is to improve the accuracy of submissions, aiming the development of scheduling algorithms which can rely on the walltime value, allowing optimizations in resource scheduling. A simple algorithm is created to verify the model's applicability, in a hipotetic reliable walltime scenario, aiming at performance enhancement. Our model presented constant gains on queue and waiting time, up to 98% in both. Moreover, the algorithm based on the model presented reductions of makespan up to 14,6%, with the worst case in approximately 9%.*

Resumo. *Grades computacionais fornecem recursos computacionais sob demanda com alocações definidas por políticas implementadas em algoritmos de escalonamento. O walltime de uma requisição designa o tempo de provisionamento estimado das tarefas, mas este é aproveitado precariamente. Este trabalho propõe o BUCMA, um modelo multialgoritmo de bonificação/punição de fila ao usuário com base em suas estimativas de walltime. Com estimativas mais precisas de walltime, entende-se que essa métrica possa ser incluída a algoritmos de escalonamento, permitindo otimizações no escalonamento dos recursos. Para validação, implementou-se um algoritmo de escalonamento simples aplicado a um cenário fictício de walltime confiável. Os resultados preliminares mostram que o modelo criado apresentou ganhos de fila e de tempo de espera constantes e de até 98%, e o algoritmo de escalonamento, reduções do makespan de até 14,6%, com pior caso em aproximadamente 9%.*

1. Introdução

Grades computacionais são infraestruturas que permitem a solicitação de recursos computacionais, a partir de um outro computador, através de um sistema de roteamento, permitindo a requisição remota desses recursos por parte de usuários da grade [Goes et al. 2005]. As grades têm se destacado devido um aumento da necessidade do uso de recursos computacionais, nos mais diversos tipos de aplicação [Batia 2017, Reis 2005]. Sua facilitação de uso para ambientes acadêmicos, a torna comum para experimentações com esse perfil. A ordem de provisionamento dos recursos é efetuada através de políticas de uso e prioridades, definidas por algoritmos de escalonamento

que reordenam as tarefas conforme essas políticas [Dubey et al. 2018, Jacob et al. 2005]. Dentre os parâmetros de submissão, têm-se a descrição dos recursos, o identificador do usuário e o *walltime*, este último trata do tempo de provisionamento estimado da requisição, segundo o usuário [Dong and Akl 2006, Colvero et al. 2005]. Observa-se que o valor do *walltime* é geralmente extrapolado, com o fim de garantir a execução completa das tarefas designadas [Goes et al. 2005]. Atualmente, alguns algoritmos de escalonamento acabam aproveitando-se dessa imprecisão, antecipando tarefas pré-agendadas, assim que surgem recursos disponíveis.

A principal motivação desta proposta é analisar o comportamento de um algoritmo de escalonamento simples, no que se refere ao uso dos recursos de uma grade, baseando-se na premissa de que os usuários fornecem valores de *walltime* precisos. Como contribuição, o trabalho apresenta o **Beneficiador do Usuário Consciente MultiAlgoritmo** (BUCMA), modelo que bonifica/pune os usuários conforme a precisão apresentada no envio dos valores de *walltime*, reordenando as filas conforme os valores obtidos para cada tarefa integrante às filas. Esse modelo atua na reconstrução das filas de tarefas que serão escalonadas pelos algoritmos disponíveis na grade de destino. O BUCMA gera ganhos no posicionamento de tarefas na fila, proporcional ao perfil de confiabilidade que o usuário apresenta. Isso implica em ganhos efetivos aos usuários que enviarem informações mais acertadas de seus *walltime*.

Para aplicação do modelo e testagem do algoritmo simples, são utilizados quatro algoritmos de escalonamento tradicionais de grades como base influenciadora do modelo e como algoritmos comparativos, respectivamente. O algoritmo *First Come First Served* (FCFS) tem sua política de prioridade na ordem de chegada das tarefas, atendendo às primeiras submetidas [Singh et al. 2017]. O *Extensible Argonne Scheduling System* (EASY) é similar, mas com o uso do preenchimento de lacunas, adiantando tarefas não prioritárias, caso haja recursos disponíveis para esta e a prioritária não possa ser executada no momento [Mu'alem and Feitelson 2001]. O MIN-MIN tem uma política que dá prioridade às tarefas com menor tempo de execução previsto – em contextos que é possível tal previsibilidade. O MAX-MIN, inverso ao MIN-MIN, tem sua prioridade nas tarefas com maior tempo de execução previsto [Sharma and Atri 2017].

Os resultados encontrados revelam eficiência tanto para o modelo BUCMA quanto para o algoritmo simples. O modelo obtém tendências de ganho de fila e de redução do tempo de espera de até 98% para os perfis mais precisos e com uma aplicabilidade mais interessante para os algoritmos FCFS e EASY. O algoritmo simples criado, por sua vez, apresentou reduções de *makespan* de até 14,6%. Este também apresentou reduções altas nas métricas de *slowdown* e tempo de espera para 3 dos 4 algoritmos com o qual foi comparado, com reduções comumente em acima de 90%, obtendo resultados menos interessantes nelas apenas quando comparado ao MIN-MIN. O restante do artigo é organizado como segue. A Seção 2 aborda os trabalhos relacionados. Na sequência, a Seção 3, descreve o BUCMA e a Seção 4 a experimentação e as análises sobre seus resultados.

2. Trabalhos Correlatos

Abordando a implementação de melhorias de algoritmos, dentro do contexto daqueles que são implementados como comparação, tem-se a aplicação do algoritmo de FCFS com uso de preempção, ponderação às requisições e com escalonamento adaptativo à carga de

entrada. Obtêm-se melhorias significativas em relação ao FCFS, apesar de resultados sensíveis às políticas de preempção escolhidas [Schwiegelshohn and Yahyapour 1998]. Também a implementação e análise da relação de desempenho entre o FCFS e o EASY, na qual o EASY relatou melhores resultados devido aos benefícios do *backfill*, medianamente em torno de 25 a 30% [Hamscher et al. 2000]. Há a aplicação do algoritmo EASY com propostas de DeepScheduler, que faz uso também de DeepLearning, e de *shutdown* dos recursos, buscando otimizar o gerenciamento de processos no escalonamento, eficiência energética e decréscimo da ociosidade dos recursos. O DeepScheduler, obteve reduções do tempo médio de espera na fila em até 26% [Casagrande et al. 2020].

Trabalhos dentro de contextos de comparativos entre algoritmos também são relevantes para determinar uma maior compreensão dentre o desempenho e concorrência entre si dos algoritmos, com foco nos usados como base comparativa. A comparação do desempenho entre os algoritmos MAX-MIN e MIN-MIN em relação ao *makespan* indica melhores resultados para o MAX-MIN nesta métrica, enquanto no MIN-MIN para o menor tempo médio de espera entre as requisições [Sharma and Atri 2017]. Foi encontrada também a aplicação do balanceamento de carga ao algoritmo MIN-MIN, obtendo diminuição de até 23% no *makespan* e uma diminuição da ociosidade no uso dos recursos disponíveis [Kokilavani and Amalarethinam 2011]. Outro trabalho similar é realizado, também obtendo uma redução próxima no *makespan* [Anousha and Ahmadi 2013]. Também foram encontrados trabalhos que tratam da combinação do MAX-MIN com o MIN-MIN, aplicando uma heurística que seleciona o mais indicado em cada cenário, obtendo resultados levemente melhores em *makespan*, mas que variam bastante conforme o cenário e os pesos atribuídos [Etminani and Naghibzadeh 2007].

Dentro do contexto abordado, já foi realizada a criação de um modelo Beneficiador de Usuários Conscientes (BUC) [Corrêa and Pillon 2020]. Este reordena as tarefas de acordo com o índice de confiança que os usuários apresentam, dando prioridade aos mais confiáveis, a partir de uma política de prioridade FCFS simples, com o fim de ser um modelo de conscientização do valor de *walltime* à longo prazo. Foram analisados o ganho de posições de filas e a redução do tempo de espera dos usuários através do modelo, ocorrido em maior tendência nos usuários de maior confiabilidade, em detrimento dos menos confiáveis Relacionado a este trabalho, falhas na construção foram a limitação do modelo a somente um algoritmo, a falta de uma análise mais extensa nas métricas de desempenho e mesmo um algoritmo simples de testagem como efeito de validação.

3. Beneficiador do Usuário Consciente MultiAlgoritmo (BUCMA)

O modelo BUCMA é fundamentado nos ganhos de fila e reduções do tempo de espera conforme o perfil do usuário. Em um ambiente de usuários despreocupados com a precisão do *walltime*, buscar uma forma de conscientizá-los, com o fim de fornecer um melhor aproveitamento, é relevante. Este modelo gera índices de confiança individuais a cada usuário de acordo com o histórico da precisão dos valores de *walltime* (Equação 1), sendo a precisão (*P*), tempo execução (T_e) e *walltime* (*W*). Conforme os valores obtidos para esses parâmetros, é gerada uma nova pontuação à cada requisição, replicando em sua nova posição para escalonamento. A motivação na aplicação com base na reincidência surge também por ser comum a prática de envio de diversas requisições por um mesmo usuário em grades computacionais [Reis 2005, Poquet 2017].

O modelo BUCMA tem o objetivo disso, a partir do índice de confiança individual de cada usuário, atribuído conforme a precisão dos valores de *walltime* das requisições passadas de cada usuário individual que submete requisições à grade. A motivação na aplicação com base na reincidência surge também por ser comum a prática de envio de diversas requisições por um mesmo usuário em grades computacionais

$$P = T_e \div W \times 100 \tag{1}$$

A partir da criação de uma variável que representa a confiabilidade de cada usuário, *i.e.*, a variável *score* (S_u), tem-se a retribuição de seu valor, a partir da precisão obtida por suas tarefas anteriores, que culminam em seu valor atual. Seu valor vai variar em um inteiro entre 0 a 100, na qual, quanto maior, maior o índice de confiabilidade que o usuário possui. O valor atual irá impactar em nos ganhos de fila e, consequentemente, o quão rápido tende a ser atendido em requisições futuras. A Tabela 1 representa a redistribuição do S_u dos usuários conforme sua precisão P obtida, isto ao fim da execução de cada uma de suas requisições. A partir de uma precisão $P > 35$, o S_u para a ser acrescido, também para dar uma margem de segurança de acerto do *walltime*, dado que ser for extrapolado, este terá sua tarefa finalizada. P menores ou iguais a 15 o S_u para a ser decrescido. Caso P seja maior que 100, é desconsiderado, devido que o RJMS para as execuções dessas tarefas e não se sabe qual foram as motivações de sua extrapolação.

Porcento	$P > 100$	$P > 80$	$P > 65$	$P > 50$	$P > 35$	$P > 25$	$P > 15$	$P > 10$	$P \geq 0$
S_u	+0	+10	+6	+3	+1	+0	-1	-2	-3

Tabela 1. Valores de S_u atribuídos às tarefas ao final de suas requisições.

Com a atribuição do S_u definida, é possível apresentar o modelo BUCMA, que utiliza, ao todo, 4 parâmetros para a obtenção da pontuação que irá reordenar as tarefas. O modelo está definido pela Equação 2, cujo a pontuação da tarefa é dada pela variável final calculada S_f. O primeiro critério é em volta do S_u do usuário e representa a sua confiabilidade. O segundo é referente ao algoritmo de base (A_b) e representa o impacto do algoritmo original aplicado ao modelo, que determina o intuito de multialgoritmo do mesmo. O terceiro é o tempo de espera atual da tarefa (*E*), considerado como critério devido aos possíveis ganhos que pode apresentar e também com intuito de evitar inanição de tarefas. O quarto critério é referente ao *walltime* da requisição (*W*), implicando em uma tentativa de um ganho imediato conforme *walltime* menor, buscando apresentar um ganho mais imediato na diminuição do *walltime* dos usuários que lançam esse parâmetro de forma tão extrapolada. Todos os critérios são normalizados, através do uso dos denominadores, se obtendo valores entre 0 e 1 para cada um dos critérios. As variáveis representam os vínculos de influência que se aplicam a cada um dos quatro critérios, sendo que cada uma destas terá seu valor variando entre 0 e 1, representando a influência sobre os critérios, e que $\alpha + \beta + \gamma + \delta = 1$.

$$S_f = \alpha \times \frac{S_u}{100} + \beta \times \frac{A_b}{Max_{A_b}} + \gamma \times \frac{E}{Max_E} + \delta \times (1 - \frac{W}{Max_W}) \tag{2}$$

Com o modelo definido, foi elaborado o plano de testes preliminar para análise do impacto dos pesos na redução do tempo na fila por perfil. Os pesos escolhidos, através dos

resultados que trouxeram ganhos mais interessante em posicionamento de fila e melhorias nas métricas de desempenho, foram: (i) 50% de influência para o critério da confiabilidade do usuário, 30% para o algoritmo de base, 0% para o tempo de espera e 20% para o *walltime* em um primeiro modelo, nomeado BUCMA5302; e (ii) modificou-se a confiabilidade para 40% e o *walltime* para 30%, obtendo-se o BUCMA4303. Dado que o índice de confiabilidade e o *walltime* revelalaram ser parâmetros mais efetivos, foram criados modelos que distinguem esses valores entre si para avaliar uma possível comparação de custo-benefício do ganho de posições associado às métricas de desempenho.

3.1. Algoritmo de escalonamento com *walltime*

A proposta de algoritmo de escalonamento para validação do modelo BUCMA atua na pré-categorização das tarefas, associando-se a um algoritmo de escalonamento que efetue o provisionamento de recursos. A pré-categorização das tarefas baseia-se na confiança no *walltime* estipulado pelo usuário, sendo este considerado um algoritmo simples. O cenário de aplicação do algoritmo parte uma precisão entre 85% a 100% no valor do *walltime* de cada requisição em relação ao seu tempo real de execução, com o objetivo de criar um cenário fictício de aplicação prévia efetiva do BUCMA no ambiente. O algoritmo também conta com o uso de *backfilling*, para preenchimento de lacunas das tarefas, *e.g.*, sempre que uma tarefa supostamente a frente na fila de execução não for possível ser executada – estando esta em estado de espera –, são analisadas as tarefas mais atrás nesta mesma fila para verificar se alguma pode ser adiantada, sem prejudicar a atual em espera.

Algoritmo 1 Abordagem ingênua baseado no BUCMA

```
1: if quantidade recursos livres > 80% then
2:     filaAtual ← tarefas ordenadas por maior walltime * recursos requisitados;
3: else
4:     filaAtual ← tarefas ordenadas por menor walltime * recursos requisitados;
5: for tarefa in filaAtual do
6:     if existem recursos livres para alocar a tarefa then
7:         Inicia tarefa e remove tarefa da filaAtual;
8:     else
9:         break;
10: primeiraTarefa ← remova a primeira tarefa da filaAtual;
11: for tarefa in filaAtual do
12:     if existem recursos para iniciar a tarefa iterada e a reserva desta não prejudica a
    primeiraTarefa then
13:         Inicia tarefa e remove tarefa da filaAtual;
```

A proposta pode ser descrita pelo Algoritmo 1, na qual se tem o pseudocódigo de aplicação do algoritmo criado. Cada evento implica na chamada do trecho de código para sua execução. O primeiro passo do algoritmo é ordenar uma fila atual de espera conforme a política de prioridade estabelecida. Após isso, será executado cada tarefa, nesta mesma ordem, dispondo recursos a esta, iniciando a tarefa e a removendo da fila, até não haver mais recursos disponíveis para a atual tarefa iterada. Quando isso ocorre, o laço de alocação é quebrado. Com isso, é pego a tarefa prioritária atual da fila, então iterado sobre o restante das tarefas ainda em fila. Enquanto houver tarefas menos prioritárias que

esta, mas com recursos disponíveis para a sua execução e que não prejudiquem a espera da tarefa prioritária, estas podem ser executadas. Assim, são também alocados recursos a essas tarefas seguintes, iniciando a tarefa e removendo-as da fila atual de espera.

Este algoritmo considera que, caso haja 80% ou mais dos recursos livres na plataforma, dá-se prioridade às tarefas de maior *walltime* multiplicadas pela quantidade de recursos requisitados. Caso contrário, a fila é reordenada pela lógica inversa quanto ao tamanho do *walltime*. Essa escolha se deve que, dado a chegada de tarefas ser aleatória e assíncrona, assim como o *walltime* neste contexto ser preciso, pode-se inferir que as tarefas com maior *walltime* são, consequentemente, maiores. Além disso, a disposição dos recursos é feito da forma do melhor disponível para o pior. Assim, parte-se de que lançar tarefas que ocupem mais recursos para processadores melhores, e vice-versa, permite obter uma maior eficiência dos tempos de execução dos recursos. O valor de 80% é uma escolha feita com base nos melhores resultados dentre várias faixas testadas empiricamente. Em outros contextos de recursos e as diferentes capacidades de processamento em cada, outro valor de divisão pode vir a ser mais interessante.

4. Avaliação Experimental

Constitui-se como base os dois modelos BUCMA definidos para cálculo do modelo e do algoritmo criado para avaliação do algoritmo simples, todos perante implementação e comparação com os quatro algoritmos base explicados. Assim, também define-se como uso a carga de entrada do sítio de Lille da GRID'5000, referente ao ano de 2020, o que totaliza 7056 requisições. Para sua análise no modelo BUCMA, foram inseridas 300 requisições, de *walltime* de 3600 segundos. Foram definidos três perfis de precisão (pouco preciso, medianamente preciso e preciso), estando cada um com S_u fixado em 0, 50 e 100, respectivamente. Para o algoritmo simples, foi utilizada a carga original cm *walltime* modificado para se adequar a uma precisão ideal – totalmente preciso, com ruído de 15%, para baixo. A plataforma de simulação de grades usada é o Batsim [Poquet 2017], em versão estendida [Casagrande 2020]. O hardware replicado no simulador é o de Lille, constituído atualmente de 39 nós, distribuídos em 4 agregados heterogêneos entre si, mas com nós internos homogêneos. A Tabela 2 representa o *hardware* replicado.

Agregado	Nós	CPU	Núcleos
Chifflet	8	2 x Intel Xeon E5-2680 v4	14/CPU
Chifflot	8	2 x Intel Xeon Gold 6126	12/CPU
Chiclet	8	2 x AMD EPYC 7301	16/CPU
Chetemi	15	2 x Intel Xeon E5-2630 v4	10/CPU

Tabela 2. Configuração do sítio Lille / GRID'5000. Adaptado de: [GRID'5000 2021]

.

4.1. Análise sobre o modelo BUCMA

Inicialmente, tem-se a avaliação da redução dos tempos de espera, conforme perfil e cada modelo BUCMA aplicado, conforme os algoritmos de base (Tabela 3). Sendo 0% o tempo original do algoritmo comparado, tem-se que -100% representa uma redução total em relação ao valor comparado, sendo qualquer valor acima de 0% representa um aumento.

Na Tabela 3, ambos os modelos reduzem de forma positiva os tempos de espera conforme o perfil, reduzindo até 97,8% e 97,3%, como nos casos do FCFS, para os modelos BUCMA5302 e BUCMA4303, respectivamente. O MIN-MIN apresenta perdas

Modelo	Perfil	FCFS	MIN-MIN	MAX-MIN	EASY
BUCMA5302	Impreciso	26,7	2053,2	8,3	20,3
	Mediano	-61,5	601,3	-7,00	-23,9
	Preciso	-97,8	-30,9	-97,1	-55,8
BUCMA4303	Impreciso	13,3	1271,4	5,0	-28,6
	Mediano	-85,8	-17,2	-86,1	-46,5
	Preciso	-97,3	-14,5	-95,3	-55,2

Tabela 3. Comparativo percentual entre modelos BUCMA aplicado aos algoritmos base em relação à redução do tempo de espera para cada perfil de usuário.

na maioria dos casos, destacando-se o aumento de 2053,2% para o perfil impreciso, no BUCMA5302. O modelo BUCMA5302 apresentou reduções melhores nos perfis mais precisos, mas piores nos imprecisos. Isso indica uma maior amplitude das reduções para o BUCMA5302, que pune ou bonifica mais conforme o perfil.

A avaliação dos ganhos de posicionamento de fila é representada em uma escala de 0% a 100%, sendo 100% representa a inclusão da tarefa na primeira posição da fila e 0% o posicionamento na última posição. Sendo assim, a expectativa em torno do ganho de fila se torna mais presente em perfis mais precisos. Para esta avaliação, foram definidos quatro perfis: $0-24$, $25-49$, $50-74$ e $75-100$. Esses perfis foram obtidos a partir da execução das tarefas e o S_u do usuário destas no momento de sua submissão na grade. Foram realizadas análises sobre os quatro algoritmos, gerando cada um gráfico *boxplot* completo para cada. Por limitação de páginas do artigo, são apresentados apenas o melhor e pior resultado.

A Figura 1 revela o desempenho do FCFS perante os ganhos de fila conforme os perfis de confiança sobre os modelos BUCMA5302 e BUCMA4303. Os resultados com o algoritmo EASY aplicado foram de consistência similar. O desempenho revela-se positivo, com a mediana com uma forte tendência de crescimento conforme as faixas de *score* aumentam. Além disso, há um considerável diferença entre os primeiros e terceiros quartis em cada faixa, indicando uma distinção de ganho de fila bem razoável entre os resultados, conforme o perfil. Na menor faixa, $0-24$, os ganhos pela mediana ficam entre 3% a 4% em ambos os modelos, com máximos na faixa dos 20% – 60% se considerar os *outliers*. Na faixa $25-49$, a mediana já passada a ficar próxima de 36% para o BUCMA5302 e 33% para o BUCMA4303. Na faixa $50-74$, se tem mediana de 62% para o BUCMA5302 e 78% para o BUCMA4303. Na última e maior faixa, indicando os usuários mais confiáveis, se tem uma mediana tendendo a 100% de ganho, sendo próximo de 97% de ganho de fila para o BUCMA5302 e 98% para o BUCMA4303. Esses ganhos indicam um julgamento, na prática, bem dentro do esperado para a aplicação do modelo.

Na análise do BUCMA5302 e BUCMA4303 sobre o algoritmo MIN-MIN (Figura 2) indica também que conforme se aumentam as faixas de *score*, o ganho de posições de fila também é acrescido, levando em consideração a mediana como fator principal de análise. Contudo, o forte *overlap* entre as regiões dificulta a análise. Todas as faixas apresentam um percentual alto na mediana, em que mesmo a faixa mais baixa, $0-24$, apresentou um ganho próximo de 67% em ambas versões para a mediana. A faixa $25-49$ apresentou um ganho positivo expressivo, com medianas de 92% para o BUCMA5302 e 90% para o BUCMA4303. A faixa $50-74$ já apresenta uma tendência de 100%, com mediana de 97% para o BUCMA5302 e 98% para o BUCMA4303. Por fim, a faixa $75-100$

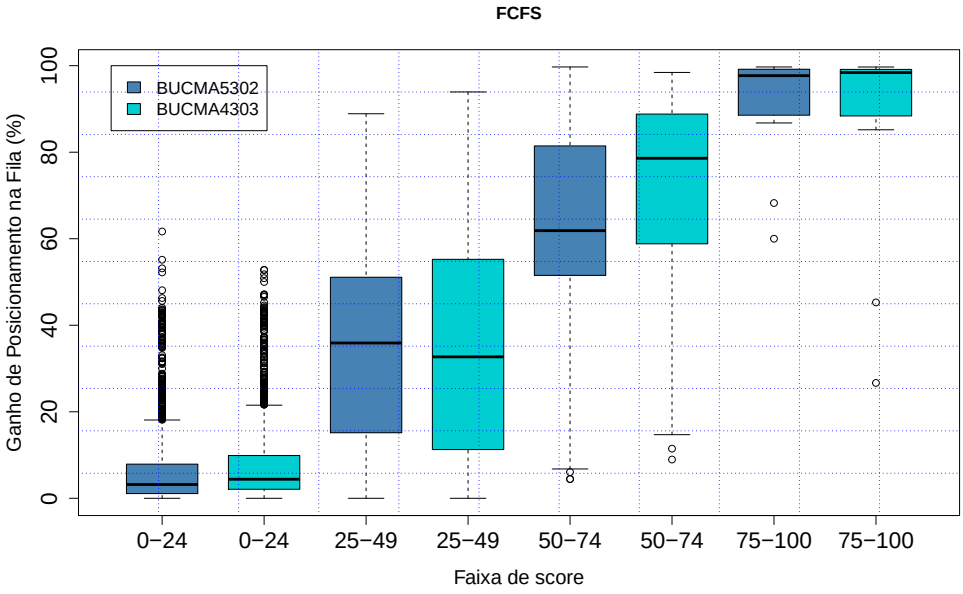

Figura 1. *Boxplot* dos modelos BUCMA5302 e BUCMA4303 aplicados ao FCFS.

apresentou uma mediana ainda um pouco acima, com variações menores e quase irrelevante entre máximos e mínimo, ficando na faixa dos 99% a 100% para ambos os modelos.

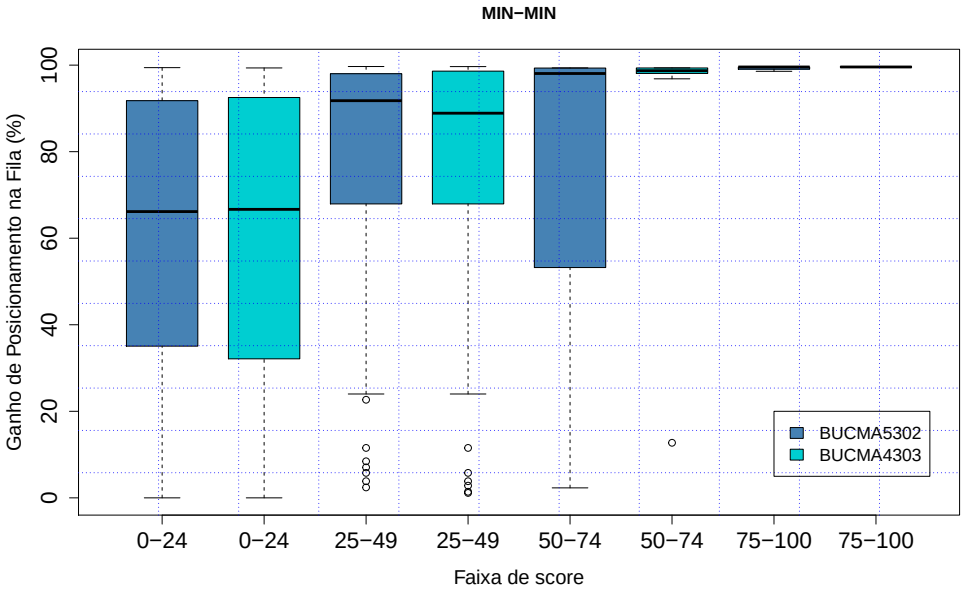

Figura 2. *Boxplot* dos modelos BUCMA5302 e BUCMA4303 aplicados ao MIN-MIN.

A principal motivação creditada para estes altos percentuais em todas as faixas na Figura 2, se deve ao formato do algoritmo MIN-MIN, que acaba acumulando tarefas grandes ao final de sua fila, fazendo com que seja recorrente ultrapassagens por tarefas menores, mesmo com baixa confiabilidade. O MAX-MIN mostrou comportamento simi-

lar. Os ganhos gerais revelaram-se positivos, implicando em boa proporção de ganho conforme os perfis. Não foram identificadas diferenças reais entre os modelos BUCMA5302 e BUCMA4303 para os testes de ganho de posicionamento de fila.

4.2. Análise sobre o algoritmo simples

A aplicação do algoritmo simples que confia no *walltime* apoiado no modelo BUCMA é avaliada nesta seção (Algoritmo 1). Conforme a Figura 3, o *makespan* obteve reduções para o algoritmo criado perante os outros quatro. O *makespan* do algoritmo simples comparado com os algoritmos FCFS, MIN-MIN e MAX-MIN foi entre 13,6% a 14,6%. Perante o EASY, que possui um *makespan* um pouco mais reduzido, foi obtida uma redução de aproximadamente 9%. Essa é a principal métrica levada em conta quando se considerada análise de melhoria de desempenho, dentro do contexto de grades computacionais.

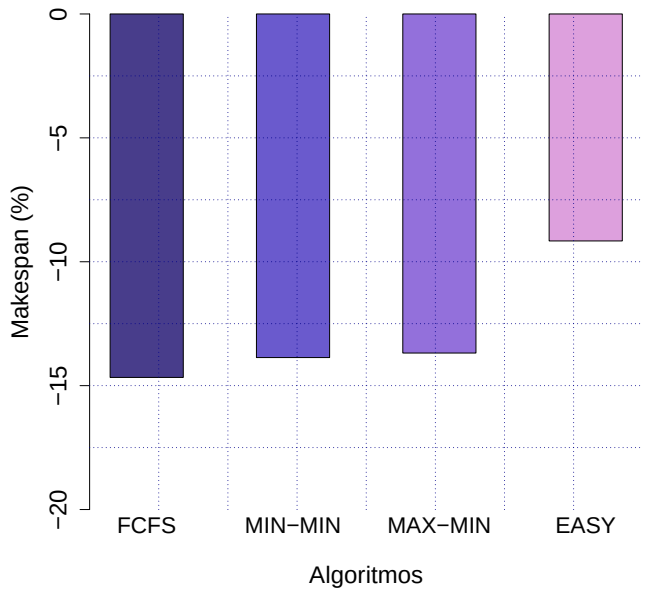

Figura 3. Percentual de tempo *makespan* do algoritmo simples em comparação aos algoritmos usados como comparação.

Nas análises de redução das métricas de tempo de espera e *slowdown*, conforme a Figura 4, os resultados constatam reduções para três dos quatro algoritmos comparados. Para o tempo de espera, foram obtidas reduções de 91%, 98% e 85% para os algoritmos FCFS, MAX-MIN e EASY, respectivamente. Para o *slowdown*, são obtidas reduções de 98%, tendência de 100% e 96% para a mesma ordem de algoritmos anteriormente citada. São reduções positivas e próximas do 100%. É válido ressaltar que esses três algoritmos não são algoritmos otimizados nessas métricas, apesar do EASY apresentar um resultado considerado interessante nelas. Por outro lado, para o MIN-MIN foi verificado um aumento de 24% para o tempo de espera e 7% para o *slowdown*, algo consequente também pelo fato do MIN-MIN ser um algoritmo otimizado nessas métricas.

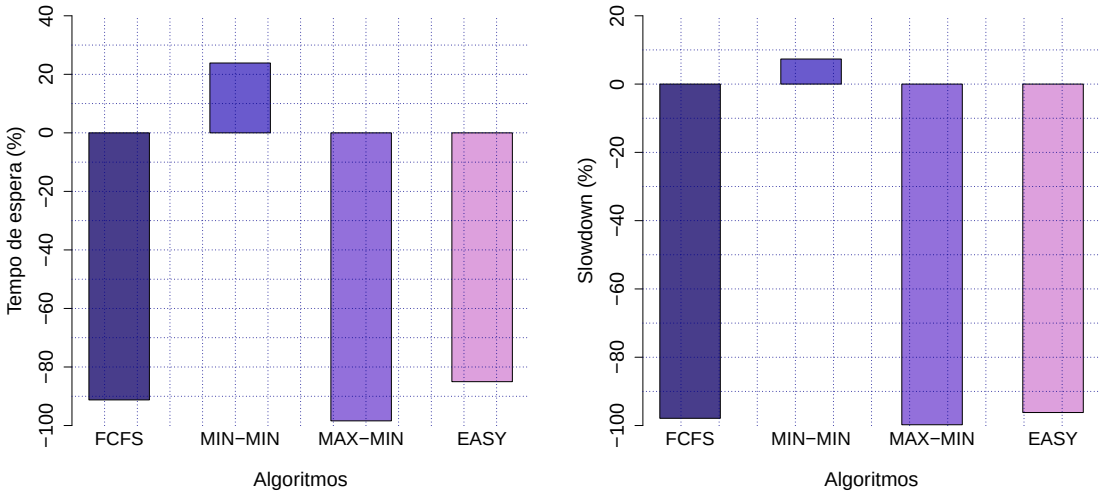

Figura 4. *Slowdown* e tempo de espera do algoritmo simples em relação aos algoritmos comparados.

Com estes resultados, é possível inferir que, partindo de um contexto ideal de aplicação, ou, ao menos, próximo, permite surtir efeitos positivos na otimização do escalonamento. Mesmo um algoritmo com uma lógica simplista, de baixa complexidade, obteve resultados interessantes, com reduções consideráveis de *makespan*, principal fator que reflete o desempenho do algoritmo, assim como reduções altas de *slowdown* e tempo de espera. O objetivo é que isso demonstre, mesmo que minimamente, que algoritmos que confiem no *walltime*, em contextos confiáveis do mesmo, permitem ganhos de desempenho. Ademais, é esperado que algoritmos mais bem elaborados permitam otimizações ainda melhores aos diferentes contextos de grades utilizadas.

5. Considerações & Trabalhos futuros

As grades computacionais são infraestruturas com expressivo potencial de alta capacidade computacional. A maximização no uso dos seus recursos figura entre os objetivos de qualquer administrador de grade. O envio de informações imprecisas por parte dos usuários dificulta as otimizações. Neste contexto, umas das principais informações incorretas é o envio do valor de *walltime* de uma requisição, na qual este tem pouco uso prático atualmente. Foi realizada a criação de um modelo que busque tornar esse valor confiável, permitindo o surgimento de algoritmos mais bem elaborados e que explorem esse valor, o que, por sua vez, pode gerar otimizações no escalonamento e nas métricas de desempenho.

O modelo criado foi o BUCMA, descrito e explicado neste artigo. Foi realizada a sua testagem através do BUCMA5302 e BUCMA4303, sendo seus resultados demonstram tendências de ganhos de fila e reduções no tempo de espera de até aproximadamente 98%. Os resultados negativos ficaram concentrados em torno do algoritmo MIN-MIN como base, indicando menor amplitude conforme os perfis de distinta confiabilidade pro ganho de fila, assim como aumento ou irrelevante redução nos tempos de espera. O modelo se demonstrou efetivo para o objetivo o qual foi proposto, apoiando-se de forma mais efetiva sobre algoritmos como FCFS e EASY.

O algoritmo simples criado também obteve resultados efetivos, com reduções de *makespan* de até aproximadamente 15%, através do algoritmo FCFS, ficando similar para o MIN-MIN e MAX-MIN. O EASY obteve a redução de *makespan* menos considerável, sendo, ainda sim, de aproximadamente 9%. Para *slowdown* e tempo de espera, as reduções foram na casa dos 90% para três dos quatro algoritmos comparados, o que implica em reduções expressivas. O MIN-MIN foi o único dos quatro algoritmos com resultados piores nessas duas métricas, obtendo aumentos de 24% e de 7% para tempo de espera e *slowdown*, respectivamente. O fato do MIN-MIN ser considerado otimizado para essas duas métricas acaba dificultando a obtenção de redução das mesmas. O desenvolvimento de algoritmos aprimorados usando a métrica do *walltime* pode implicar em retornos ainda mais positivos.

Como trabalhos futuros, uma avaliação da aplicação do modelo BUCMA em um cenário real, ao longo do tempo, para avaliar uma possível real conscientização é criada com base nas políticas de bonificação e punição. Além disso, a análise de métricas de redução do consumo energético para os algoritmos criados, assim como similares, também é interessante e pode servir como um motivador para adoção do BUCMA e outros modelos similares.

Agradecimentos

Os autores agradecem a Fundação de Amparo a Pesquisa de Santa Catarina (FAPESC), a GRID'5000 e o Laboratório de Processamento Paralelo Distribuído (LabP2D), do CCT/UDESC.

Referências

Anousha, S. and Ahmadi, M. (2013). An improved min-min task scheduling algorithm in grid computing. In *GPC 2013*, pages 103–113.

Batia, M. K. (2017). Task scheduling in grid computing: A review. In *Advances in Computational Sciences and Technology*, pages 1707–1714. ISNN, Research India Publications.

Casagrande, L. (2020). batsim-py 1.0.3. https://pypi.org/project/batsim-py/1.0.3.

Casagrande, L., Koslovski, G., Miers, C. C., and Pillon, M. (2020). DeepScheduling: grid computing job scheduler based on deep reinforcement learning. In *The 34-th International Conference on Advanced Information Networking and Applications (AINA-2020)*.

Colvero, T. A., Dantas, M. A. R., and Cunha, D. P. d. (2005). Ambientes de *Clusters* e *Grids* computacionais: Características, facilidades e desafios. *I Congresso Sul Brasileiro de Computação*.

Corrêa, G. and Pillon, M. A. (2020). BUC: Beneficiador de usuários conscientes em grades. *18ª Escola Regional de Redes de Computadores*.

Dong, F. and Akl, S. G. A. (2006). Scheduling algorithms for grid computing: State of the art and open problems. In *School of Computing*, page 55, Kingston, Ontario.

Dubey, K., Kumar, M., and Sharma, S. C. (2018). Modified HEFT algorithm for task scheduling in cloud enviroment. In *6th International Conference on Smart Computing and Communications*, volume 125, pages 725–732, Kurukshetra, India. Procedia Computer Science.

Etminani, K. and Naghibzadeh, M. (2007). A min-min max-min selective algorihtm for grid task scheduling. In *2007 3rd IEEE/IFIP International Conference in Central Asia on Internet*, pages 1–7.

Goes, L. F. W., Neto, D. O. G., Ferreira, R., and Cirne, W. (2005). Computação em grade: Conceitos, tecnologias, aplicações e tendências. In *Escola Regional de Informática de Minas Gerais*.

GRID'5000 (2021). Grid5000. https://www.grid5000.fr/w/Hardware.

Hamscher, V., Schwiegelshohn, U., Streit, A., and Yahyapour, R. (2000). Evaluation of job-scheduling strategies for grid computing. In *Grid Computing — GRID 2000*, volume 1971, pages 191–202.

Jacob, B., Brown, M., Fukui, K., and Trivedi, N. (2005). *Introduction to Grid Computing*. IBM. https://www.redbooks.ibm.com/redbooks/pdfs/sg246778.pdf.

Kokilavani, T. and Amalarethinam, D. I. (2011). Load balanced min-min algorithm for static meta-task scheduling in grid computing. In *International Journal of Computer Applications*, volume 20. DOI:10.5120/2403-3197.

Mu'alem, A. W. and Feitelson, D. G. (2001). "Utilization, predictability, workloads, and user runtime estimates in scheduling the IBM SP2 with backfilling". In *IEEE Transactions on Parallel and Distributed Systems*, volume 12, pages 529–543.

Poquet, M. (2017). *Approche par la simulation pourla gestion de ressources*. PhD thesis, Université Grenoble Alpes. htpps://tel.archives-ouvertes.fr/tel-01757245.

Reis, V. Q. d. (2005). Escalonamento em *grids* computacionais: estudo de caso. Master's thesis, USP, São Carlos. https://www.teses.usp.br/teses/disponiveis/55/55134/tde-18092006-115903/en.php.

Schwiegelshohn, U. and Yahyapour, R. (1998). Analysis of first-come-first-serve parallel job scheduling. *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms*.

Sharma, N. and Atri, S. T. S. (2017). A comparative analysis of min-min and max-min algorithms based on the makespan parameter. *International Journal of Advanced Research in Computer Science*, 8(3).

Singh, A. B., Bhat, S., Raju, R., and D'Souza, R. (2017). A comparative study of various scheduling algorithms in cloud computing. *American Journal of Intelligent Systems*, 7(3):68–72.