Análise de estratégias de escalonamento para aplicações moldáveis com deadline

Henrique Klôh[1], **Vinod Rebello**[1], **Bruno Schulze**[2]

[1]Instituto de Computação – Universidade Federal Fluminense (UFF)
Niterói – RJ – Brasil

[2]ComCiDiS – Laboratório Nacional de Computação Científica (LNCC)
Petrópolis – RJ – Brasil

{hkloh,vinod}@ic.uff.br, schulze@lncc.br

Abstract. *The amount of resources necessary for the execution of an application is generally defined by the user. If the application is rigid, this decision can be easy since it can only be executed with fixed number of resources. However, for moldable applications, users may wish to take other considerations in to account when making their choice: resource availability; required execution time and/or cost; etc. One additional advantage for a moldable application is the possibility to start its execution sooner with fewer resources instead of waiting for the chosen number of the resources to become available. In relation to this problem, this paper proposes the analysis of moldable applications with relation to performance based on the percentage of total resources requested as an alternative to maximize the use of these resources, ensuring the pre-established priority order and also an alternative to optimize deadline related metrics. Another contribution presented in this work is the execution of experiments with different queue types and workloads to assess the performance of the proposed approach for each environment. The experiments present satisfactory results for moldable applications using the proposed approach combined with Backfilling.*

Resumo. *A quantidade de recursos necessários para a execução de uma aplicação é, em geral, definida pelo usuário que deseja executá-la. Esta demanda por recursos pode ser definida através de experimentos prévios a fim de identificar o melhor speedup para esta aplicação podendo ser fixa ou moldável. Uma aplicação moldável tem como vantagem adicional a possibilidade de começar a executar mais cedo com menos recursos do que esperar pela liberação da quantidade de recursos definida pelo usuário. Com base neste problema, este trabalho, propõe a análise de aplicações moldáveis com desempenho baseado na porcentagem do total de recursos solicitados como uma alternativa para maximizar a utilização destes, garantindo a ordem de prioridade preestabelecida e também como uma alternativa para otimizar métricas relacionadas ao deadline. Outra contribuição presente neste trabalho é a realização de experimentos variando os tipos de fila e cargas de trabalho a fim de avaliar o comportamento da abordagem proposta em relação a cada ambiente. Os experimentos realizados apresentaram resultados satisfatórios para aplicações moldáveis quando utilizando a abordagem proposta junto com Backfilling.*

1. Introdução

A tendência atual no projeto de servidores multicores utilizados em ambientes de HPC (*High Performance Computing*) está motivando provedores de aceitar graus cada vez mais consolidados em poucos servidores. Porém a utilização de forma eficiente destes recursos, estando agora concentrados muitas vezes em apenas um ou poucos servidores, e a crescente demanda de usuários de HPC tem nos levado a necessidade do compartilhamento deste tipo de ambiente. Neste tipo de ambiente compartilhado, os recursos não encontram-se dedicados a execução de uma aplicação, isto é, existem aplicações que já estão utilizando os recursos e o escalonador deve decidir se a aplicação que está no inicio da fila pode ser executada com os recursos que estão disponíveis ou se a mesma deve aguardar pela liberação de mais recursos para ser executada. Entretanto, caso os recursos não sejam suficientes e a aplicação necessite esperar para ser executada, estes recursos livres permanecerão ociosos até que os requisitos da aplicação sejam satisfeitos. Maximizar a utilização dos recursos e garantir a execução de todas as aplicações recebidas é o problema a ser tratado pelo escalonador.

Para atender a todas as aplicações recebidas com suas diferentes prioridades, várias políticas de escalonamento *Fair-Share* tem sido propostas visando o compartilhamento dos recursos de forma justa. Porém definir uma política de escalonamento que seja justa é relativo, e geralmente depende do ponto da vista. A política pode ser "justa" com um tipo de usuário do ambiente ou classe de aplicação e não com outro dependendo dos seus objetivos, ou, como é mais comum, favorecendo o provedor por tentar melhor a utilização a custo da redução do desempenho para aplicações dos usuários. Desta forma é difícil afirmar o quão *"fair"* uma política *Fair-Share* acaba sendo de fato.

Já a maximização da utilização dos recursos, geralmente, é trabalhada através da técnica de *Backfilling*. Esta permite que, uma vez que não há recursos disponíveis o suficientes para executar o *job* do inicio da fila de espera, seja verificado se existe um *job* com menos prioridade a esse que pode ser executado com os recursos disponíveis naquele momento. Porém esta técnica pode gerar o atraso na execução do *job* que já estava aguardando pela liberação dos recursos, dependendo de como a fila for implementada. Uma solução que é utilizada para minimizar esse atraso é só permitir um *job* posterior passar a frente quando o tempo de execução deste impactar o mínimo possível em relação ao tempo de espera pelos recursos que faltam para a execução do *job* que está no inicio da fila[Li et al. 2014].

Tanto o *Fair-Share* como a utilização de técnicas como *Backfilling*, ou até mesmo a utilização de ambas em conjunto, dependem, muitas vezes, da prioridade da aplicação ou do usuário e do perfil de consumo de recursos da aplicação a ser executada para saber se há recursos para atendê-la. Essa quantidade de recursos necessários para a execução da aplicação é determinada, em geral, pelo usuário no momento da submissão da aplicação. Este determina a quantidade de recursos com base nos dados de entrada a serem utilizados por sua aplicação em execuções anteriores. Na prática, esse valor é determinado pela quantidade de recursos onde a aplicação ofereceu o melhor *speedup*. Porém a variação do *speedup* da aplicação não é necessariamente linear mediante ao aumento da quantidade de recursos a serem utilizados. Pelo contrário, a tendência é que uma aplicação tenha um crescimento do *speedup* inicialmente e depois, vá reduzindo o seu crescimento até chegar um momento no qual o problema, ou a sua entrada, se tornam desproporcionais

para a quantidade de tarefas geradas. Sendo assim, o tempo para a geração e distribuição destas tarefas se torna maior do que o ganho obtido pela distribuição do processamento, acarretando em uma perda de desempenho, mesmo com a utilização de mais recursos.

A demanda por recursos para as aplicações, como definida por Feitelson em [Feitelson et al. 1997], pode ser rígida, moldável, evolutiva e maleável. Uma aplicação que possui uma quantidade fixa de recursos é rígida e não pode ter esse valor alterado. Já para uma aplicação moldável, como é o caso de grande parte das aplicações paralelas existentes atualmente, há a possibilidade da mesma ser executada variando com mais ou menos recursos do que o estipulado inicialmente. As aplicações evolutivas e maleáveis são semelhantes às aplicações moldáveis, porém permitem a variação da quantidade de recursos que estão sendo utilizados pela aplicação em tempo de execução. O que difere um tipo do outro é que no caso das evolutivas essa variação ocorre por uma demanda da aplicação, esta solicita a variação dos recursos. Já para aplicações maleáveis, as aplicações respondem a disponibilidade do sistema, o mesmo oferece recursos para aplicação mediante a disponibilidade e a necessidade da mesma. Para a utilização de abordagens como a de aplicações evolutivas ou maleáveis, é necessária, muitas vezes, uma reimplementação das aplicações comumente utilizadas para que elas se tornem autogerenciáveis. Aliado a este problema estas abordagens são mais complexas e mais onerosas. Porém a solução proposta para aplicações moldáveis pode ser igualmente aproveitada por aplicações evolutivas ou maleáveis.

Desta forma, podemos perceber que a quantidade de recursos para a execução de uma aplicação, em alguns casos, será fixo, pré-determinado que deve ser respeitado pelo sistema. Entretanto, há outras aplicações onde a variação da quantidade de recursos ofertados não trará um impacto negativo no seu funcionamento, acarretando somente em um tempo de execução maior.

Com base no problema de escalonamento de ambientes de HPC compartilhados, e na questão referente à quantidade de recursos necessários para a execução de uma aplicação, este trabalho propõe a análise do impacto de aplicações moldáveis e rígidas em relação às abordagens de escalonamento clássicas. Aliada a esta análise é proposta a utilização de *Backfilling* com aplicações moldáveis utilizando uma abordagem baseado na porcentagem de recursos disponíveis em relação ao total solicitado, como uma alternativa para manter a prioridade de *jobs* determinada pela fila utilizada, e garantir a maximização da utilização dos recursos do ambiente de HPC. Esta técnica de porcentagem é utilizada uma vez que não há recursos para atender ao *job* que está aguardando no inicio da fila, então é proposta uma porcentagem, ou seja, com os recursos disponíveis restantes, é ofertado uma parte do total de recursos que o *job* necessita para executar para que o mesmo possa ser executado imediatamente ao invés de aguardar indefinidamente pela liberação dos demais recursos. Através destas abordagens o presente trabalho visa avaliar o impacto de diferentes abordagens de escalonamento considerando que as aplicações podem ser rígidas ou moldáveis e avaliar estas abordagens através de diferentes métricas, a fim de identificar o impacto das abordagens e das métricas selecionadas no objetivo final do usuário.

Na Seção 2 são discutidos alguns trabalhos focados no problema de escalonamento de tarefas moldáveis e compartilhamento de recursos. Na Seção 3, é apresentado o problema a ser estudado neste trabalho. Seção 4 descreve os experimentos realizados

com base no problema descrito e nas possíveis soluções apresentadas. Por fim, é apresentado na Seção 5 a conclusão obtida através dos resultados dos experimentos e os trabalhos futuros propostos também com base nestes resultados.

2. Trabalhos Relacionados

Trabalhos como [Cirne and Berman 2002] e [Liu and Weissman 2015] indicam o interesse continuo na estratégia de *jobs* moldáveis, onde um *job* pode receber mais recursos para ser executado de forma mais rápida e liberar recursos o quanto antes para liberar espaço para outros *jobs* ou pode ter sua quantidade de recursos reduzida para se adaptar a um fragmento de espaço disponível aumentando proporcionalmente o seu tempo de execução. O trabalho de [Utrera et al. 2012], nos revela uma abordagem semelhante variando o tipo de filas utilizados entre FCFS, *Easy backfilling* e o que o autor chama de FCFS-*malleable* que ainda, segundo o autor, permite reduzir ao máximo a fragmentação e iniciar o máximo de *jobs* o quanto antes sem se preocupar com o tempo que levaram para terminar. O algoritmo proposto apresentou uma média de tempo de resposta, tempo de espera e de *slowdown* melhores do que o algoritmo *Easy Backfilling*.

Em [Huang et al. 2013a] é apresentado um trabalho que trata do problema de escalonamento de aplicações moldáveis em ambientes HPCaaS. Neste são apresentadas duas abordagens de escalonamento uma com base no tempo de execução da aplicação e outra sem o conhecimento desta informação. Segundo o autor, ambas as propostas ofereceram melhoras significantes as demais abordagens comparadas quanto a média de tempo de *turnaround*, segundo o mesmo as abordagens chegaram a ser 78% e 89% superiores as demais.

Já o trabalho apresentado em [Dutot et al. 2005] relata uma abordagem de escalonamento de aplicações moldáveis do tipo BSP (*Bulk Synchronous Parallel*). Neste trabalho a utilização do BSP é o que permite os *jobs* se tornarem moldáveis. Segundo o autor esta proposta tem por base a variação do tempo da aplicação, caso os recursos não sejam suficientes em termos de CPU, de forma que esta seja o resultado da divisão do número de processadores requeridos para execução da aplicação, dividido pelo máximo de processadores livres no recurso, multiplicado pelo tempo estimado de execução da aplicação. Segundo os mesmos esta abordagem tem por objetivo minimizar o *makespan* da aplicação.

O trabalho presente em [Jansen 2012] nos remete a um algoritmo de aproximação para o problema de escalonamento de aplicações moldáveis e não moldáveis em um mesmo conjunto de recursos. Entretanto, no trabalho apresentado em [Wu et al. 2015], é apresentado tal problema focando em ter como métrica principal de desempenho a média de turnaround do conjunto das aplicações. Neste trabalho é proposto o algoritmo HRF (*highest revenue first*) que tem por objetivo maximizar o rendimento dos recursos no menor intervalo de tempo possível para reduzir a média de *turnaround* das aplicações.

Trabalhos como o apresentado em [Hunold 2015], difere dos anteriormente citados pois levam em consideração o *speedup* da aplicação para a variação ou não dos recursos. Pois é comprovado, através de experimentos e diversos trabalhos, que cada aplicação se comporta de uma forma diferente e existe uma grande diferença entre a teoria apresentada anteriormente de aplicações moldáveis, e a prática. Na prática, o que podemos ver são variações não lineares no desempenho de acordo com a variação dos

recursos, por exemplo, para grande parte das aplicações, oferecer o dobro de recursos não garantirá que ela obtenha um desempenho duas vezes melhor. Em [Huang et al. 2013b] os autores apresentam uma abordagem para *jobs* moldáveis onde o escalonamento de cada um é baseado no seu *speedup* e na carga de trabalho.

Com relação ao fator *deadline* da aplicação, o trabalho [Chen and Matis 2013] apresenta uma política de escalonamento que organiza a fila de entrada do escalonador dando prioridade aos *jobs* que têm o vencimento dos seus prazos mais próximos. Porém, este tenta reduzir o impacto da reordenação dos *jobs* na eficiência do escalonamento em si. Em [Yang et al. 2012] o autor aborda o mesmo problema através de algoritmos genéticos visando a minimização da soma dos tempos de atraso de cada *job* mais os tempos dos *jobs* que executaram de forma antecipada ao seu prazo. Com relação a tarefas moldáveis com prazo, o trabalho apresentado em [Saule et al. 2010] propõe um algoritmo de escalonamento que oferece bons resultados, segundo o autor, para tarefas de tamanho pequeno.

Pelo que podemos identificar nos trabalhos citados, muitos dos relacionados ao escalonamento de aplicações moldáveis tem como métrica para a análise do desempenho apenas a média do *turnaround* e estas aplicações não possuem prazos para terminar a sua execução como *deadlines* ou *due dates*, uma métrica cada vez mais relevante dado o interesse crescente em computação em nuvem, qualidade de serviço e *Service Level Agreements*. Desta forma, neste trabalho é proposto o estudo do uso de aplicações moldáveis com prazos para suas execuções considerando diferentes abordagens de escalonamento e diferentes métricas para a avaliação dessas abordagens.

3. Problema

O escalonamento de *jobs* em ambientes compartilhados depende que haja a liberação dos recursos necessários para a execução do primeiro *job* da fila do escalonador para que esse possa ser alocado. Porém, em alguns ambientes, é comum que haja recursos disponíveis, mas que não sejam suficientes para a execução do *job* em questão. Para evitar que recursos fiquem ociosos, surgiram técnicas que até hoje são estudas e aprimoradas como, por exemplo, o *Backfilling*.

Com relação ao *job* que está aguardando a liberação dos recursos, algumas considerações são importantes de serem analisadas, tais como: quem define a quantidade de recursos que a aplicação necessita para executar? Esse valor é constante? Esse valor tem que ser fixo ou poderia ser variado mediante a oferta de recursos? Não seria melhor executar com um pouco menos de recursos do que ficar aguardando a liberação dos recursos que faltam? Qual é o objetivo a ser otimizado? Qual abordagem de escalonamento deve ser utilizada para este objetivo? Qual métrica deve ser utilizada para avaliar essa abordagem?

De uma forma geral, a quantidade de recursos necessários é definida pelo usuário ao identificar a melhor relação de *speedup* para a aplicação, considerando que o mesmo conheça de fato a sua aplicação. Porém, a ideia apresentada neste trabalho é que a variação deste valor estabelecido pelo usuário para a execução da aplicação pode apresentar resultados satisfatórios. Esta variação das necessidades de recursos de uma aplicação para que a mesma possa se adaptar a quantidade de recursos disponíveis, torna esta uma aplicação moldável.

Ao analisarmos a variação do tempo de execução de uma aplicação paralela com

um mesmo conjunto de dados, variando a quantidade de recursos ofertados, é possível perceber que, para cada aplicação, a variação dos recursos, seja para mais ou para menos, pode ter impactos distintos. Por exemplo, em aplicações baseadas em matrizes de tamanhos fixos, é comum ter uma redução linear do tempo até que em um determinado momento, e mesmo aumentando os recursos, o ganho de tempo pode ser pouco significativo. Se continuar aumentando a quantidade de recursos para a mesma matriz, a tendência é que o tempo comece a piorar gradativamente. Isso se dá pois o paralelismo chegou ao nível onde a aplicação está perdendo mais tempo com a transferência dos dados e com a comunicação, do que com o processamento em si.

Quanto ao critério de otimização do escalonador, cada usuário tem um objetivo que pode ser: reduzir o custo, ou reduzir o *turnaround*, ou atender o *deadline* dos *jobs* por exemplo. De uma forma geral a abordagem de escalonamento, que deveria ser de acordo com os critérios do usuário, é estabelecida pelo provedor dos recursos sem se preocupar em atender os diferentes perfis de usuários. Para avaliar essas abordagens é necessário estabelecer qual métrica será utilizada para avaliar a abordagem, por exemplo, em geral, utilizam-se métricas como o *turnaround*. Porém existem diferentes métricas onde cada qual pode representar em um ou outro caso um objetivo do usuário ou critério de escalonamento. Com base neste problema apresentado, o presente trabalho visa apresentar um conjunto de experimentos que avalie diferentes abordagens com e sem o uso de variação, e avalie estas abordagens através de diferentes métricas.

4. Experimentos

Estes experimentos têm por objetivo analisar o comportamento do escalonamento utilizando variações dos recursos necessários para a execução da aplicação quando os recursos disponíveis não são suficientes para a execução do *job*.

Para a realização do conjunto de testes foi implementado um simulador em Java que simula o processo de escalonamento de um ambiente de *job shop scheduling*. O simulador Simula um conjunto de recursos com capacidade de núcleos N e, aloca os *jobs* de acordo com a abordagem de escalonamento testada, quando pronto o mesmo é retornado. Este simulador recebe como entrada dados de *jobs* obtidos de *workloads* [Lublin and Feitelson 2003]. Estes *workloads* são amplamente conhecidos por servirem de base para simulações de ambientes de *Grids*, *Clouds* e HPC. Dentre os disponíveis, foi selecionado o *workload* Zewura e deste arquivo foram consideradas as seguintes informações de cada *job*:

- Id do *job*: identificador do *job*;
- Tempo de execução: ou tamanho do *job*, define quanto tempo aquele *job* demora para ser executado com a quantidade de recursos necessários para a sua execução;
- Quantidade de núcleos: é a quantidade de núcleos ideal, definida pelo usuário, para a execução do *job*;
- *Due date*: é o prazo estabelecido para a execução de *job*, este prazo deve ser sempre maior que o seu tempo de execução. Este parâmetro não existia inicialmente no *workload* e foi gerado e adicionado ao mesmo. O *due date* é gerado randomicamente de forma a ser entre 3 e 6 vezes maior que o tempo de execução. Este cálculo foi gerado após vários experimentos a fim de encontrar um valor intermediário de forma que os prazos não fossem nem sempre atendidos e nem sempre descumpridos.

Apesar do *workload* informar diferentes tempos de chegada para cada *job*, esta informação não foi utilizada. Na simulação, foi considerado que todos os *jobs* possuem praticamente o mesmo tempo de chegada de forma a causar a sobrecarga dos recursos até que todos os *jobs* sejam executados. Cada *job* que é carregado é alocado diretamente na fila para execução. Desta forma, em uma fila FIFO, a ordem de chegada é determinada pela ordem em que os *jobs* foram carregados para a fila.

Foram executados 400 *jobs* em um ambiente com a disponibilidade de 100 núcleos para que fosse possível a sobrecarga do ambiente. Desta forma existem 100 núcleos no ambiente, todos com a mesma capacidade de processamento. Quanto ao *workload*, este é formado por *jobs* pequenos e que consomem poucos recursos. Em média, cada *job* precisa de apenas 5 núcleos para executar. Em comparação com um ambiente real, este *workload* representa um ambiente onde os usuários possuem aplicações de pequeno porte, cada qual com um percentual de consumo que representa menos de 3% dos recursos disponíveis.

Durante a simulação, foi considerada apenas a quantidade de núcleos livres como limitador para saber se há ou não condições de executar um determinado *job* em um dado momento. Neste experimento, como o objetivo é verificar o comportamento das filas e das métricas em cada situação, partimos do principio que sempre há a disponibilidade dos demais tipos de recursos no servidor. Para a execução de um *job*, basta que ele seja o próximo da fila e haja núcleos disponíveis suficientes para a sua execução. Para cada abordagem, serão realizadas simulações com e sem a possibilidade de variação dos recursos necessários e com ou sem o uso de *Backfilling*, foram utilizados os seguintes tipos de fila em cada abordagem:

- FIFO (*First In First Out*): o primeiro *job* a entrar na fila será o primeiro a sair;
- EDF (*Earliest deadline first*): a fila será ordenada por *deadline* em ordem crescente. *jobs* que têm o prazo de execução mais próximo têm maior prioridade.
- SJF (*Shortest Job First*): a fila será ordenada pelo tempo de execução em ordem crescente. Dando maior prioridade aos *jobs* que são menores.

A fim de simular a perda de desempenho causada pela redução dos recursos utilizados na execução, será aplicado o percentual de perda para cada aplicação com base nos valores pré-estabelecidos como mostrado na Tabela 1. A tabela foi montada com base na extrapolação (limite superior) do aumento do tempo em função da variação dos recursos com base em experimentos realizados com uma aplicação paralela (o *benchmark* LINPACK).

Percentual de Recursos	100%	80%	70%	60%	50%	40%
Tempo Normalizado	1	1.3	1.5	1.8	2.1	2.5

Tabela 1. Tabela de variação.

Como métricas para avaliação de cada técnica de escalonamento foram utilizados os seguintes dados coletados durante cada teste:

- Quantidade de *jobs* atrasados: é a quantidade de execuções que terminaram após o *deadline*;

- Somatório dos atrasos: é a soma do quanto cada *job* que não cumpriu o *deadline* extrapolou do *deadline*. É representada em Milissegundos.
- Tempo de *Turnaround*: é o tempo médio (aqui em milissegundos) que um *job* leva desde o momento que é enviado para ser executado (entrada na fila) até o fim da execução;
- Tempo total de simulação: é o tempo em milissegundos que a simulação leva desde carregar e alocar os *jobs* na fila até que todos os *jobs* tenham sido executados.
- *Turnaround step by step*: permite identificar a média de *turnaround* dividindo os *jobs* em janelas de execução. Esta métrica calcula a média de *turnaround step by step*, de 10 em 10 *jobs* por ordem de chegada. Desta forma conseguimos a média de *turnaround* a cada *step*.

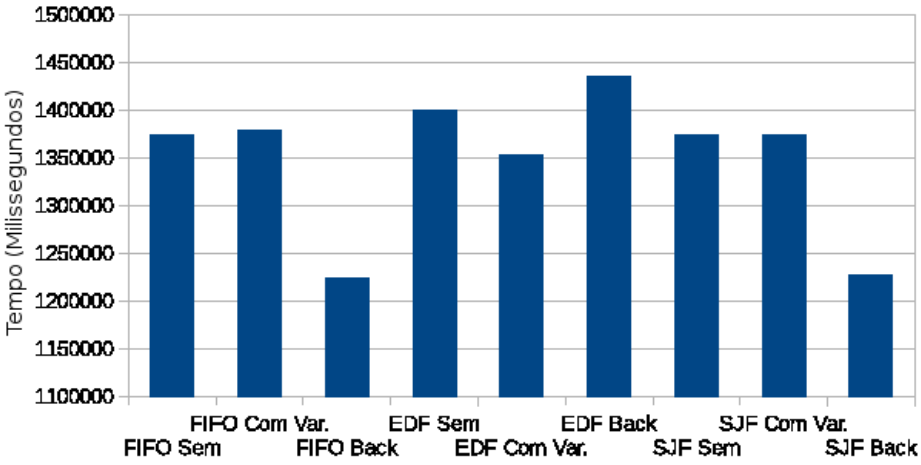


Figura 1. Tempo total de escalonamento e execução dos *jobs*.

A Figura 1 apresenta os resultados referentes a métrica de Tempo Total de simulação para as abordagens testadas. Neste caso a variação foi utilizada quando não há recursos suficientes para atender ao *job* que está no inicio da fila aguardando para ser executado. Os resultados para esta métrica foram os que apresentaram menor variação em relação às demais utilizadas. Os melhores resultados foram obtidos nas abordagens FIFO e SJF, ambos com *Backfilling*, com tempos 11% inferiores aos seus casos bases (FIFO e SJF sem *Backfilling* ou variação). Para as abordagens com EDF, é possível perceber que o uso de variação acarretou em uma leve melhora (4% em relação ao EDF sem variação) e o EDF com *Backfilling* obteve o pior desempenho entre todas as abordagens testadas.

O gráfico apresentado na Figura 2 contempla os resultados referentes a quantidade de *jobs* atrasados para cada abordagem. É interessante percebermos, que para cada tipo de fila a abordagem com variação apresenta sempre um caso intermediário entre a abordagem básica e a abordagem com o uso de *Backfilling*. Isso acontece exatamente por ser uma abordagem que procura respeitar a ordem da fila oferecendo uma alternativa para a execução do *job* em espera através da variação da quantidade de recursos necessários para a sua execução. Porém a técnica de *Backfilling* tem se mostrado mais eficiente para esta métrica, por sacrificar os *jobs* que ficam em espera, por necessitarem de mais recursos,

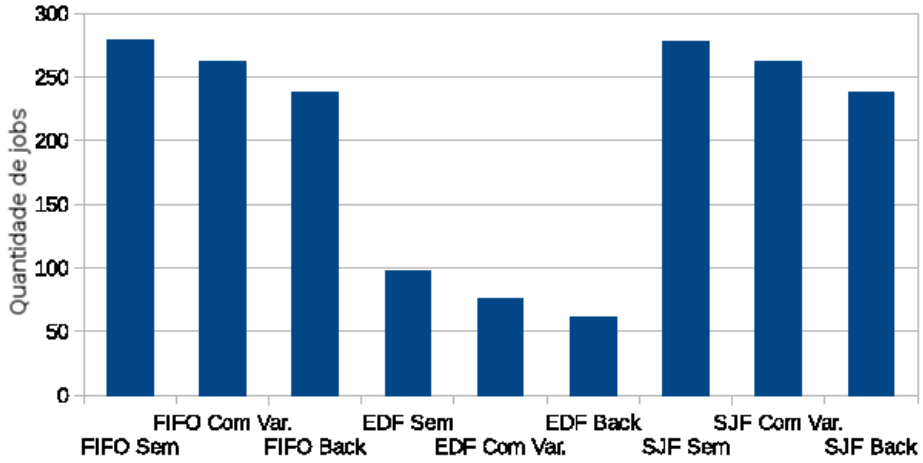


Figura 2. Quantidade de *jobs* que passaram do *deadline*.

para atender *jobs* posteriores a estes quando os recursos oferecidos naquele momento são suficientes para a sua execução. Para esta métrica as abordagens com EDF apresentaram os melhores resultados, onde a utilização da variação e o uso de *Backfilling* permitiram uma redução de 22% e 36%, respectivamente, em relação ao EDF sem estas técnicas.

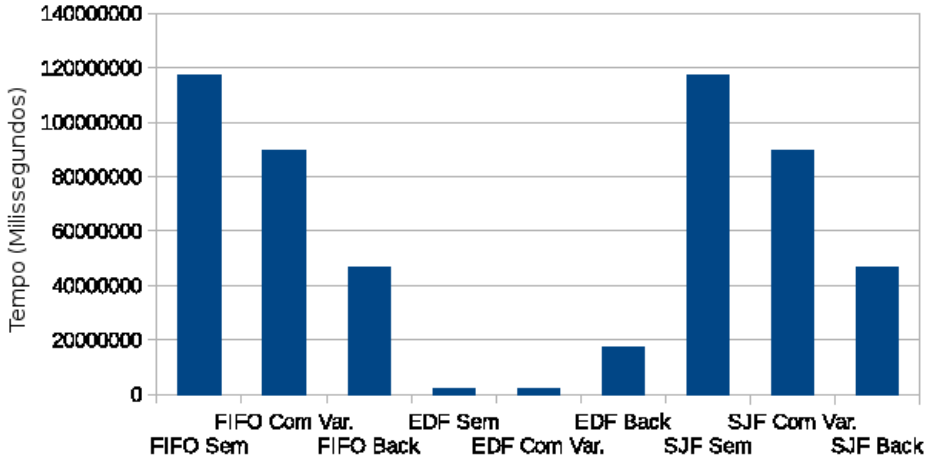


Figura 3. Somatório dos atrasos dos *jobs*.

Na Figura 3, é apresentado o gráfico com os resultados referentes ao somatório dos atrasos para cada abordagem. No gráfico podemos perceber que o comportamento das abordagens com FIFO e SJF foram semelhantes ao ocorrido para a métrica referente à quantidade de *jobs* atrasados. De igual forma, os melhores resultados foram obtidos nas abordagens EDF. Porém, a abordagem EDF com *Backfilling* foi consideravelmente pior que o EDF sem esta técnica, chegando a apresentar um tempo 7 vezes maior. Isso se deu, como mostrado no experimento 1, devido ao grande atraso nos *jobs* que necessitam de mais recursos, causado pelo uso de *Backfilling* com filas EDF.

A Figura 4 apresenta em destaque os resultados de EDF com e sem a variação. Através deste gráfico é possível perceber que a utilização da variação possibilitou uma melhora no desempenho, reduzindo o somatório dos atrasos em 5%. Apesar da diferença ser relativamente pequena, o uso de variação se mostrou vantajoso para todas as métricas em relação aos seus casos bases.

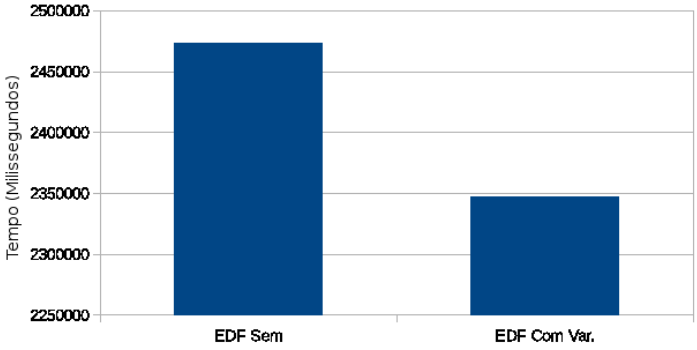


Figura 4. Somatório dos atrasos dos *jobs* para as abordagens com EDF.

Em relação à média de *Turnaround*, apresentada na Figura 5, foi identificada a mesma relação que para as métricas anteriores, onde o uso de variação permitiu obter um resultado intermediário, otimizando esta métrica em relação ao caso base para cada tipo de fila e respeitando a ordenação dos *jobs* realizada pelas filas. Porém, os melhores resultados foram obtidos, mais uma vez, pelas abordagens com o uso de *Backfilling* com tempos 52% menores para FIFO e SJF e 37% para EDF, em relação às suas abordagens sem o uso da mesma. O melhor resultado com variação foi obtido com a fila EDF com tempos 9% menor que o caso base EDF e praticamente iguais as abordagens FIFO e SJF, ambos com *Backfilling*.

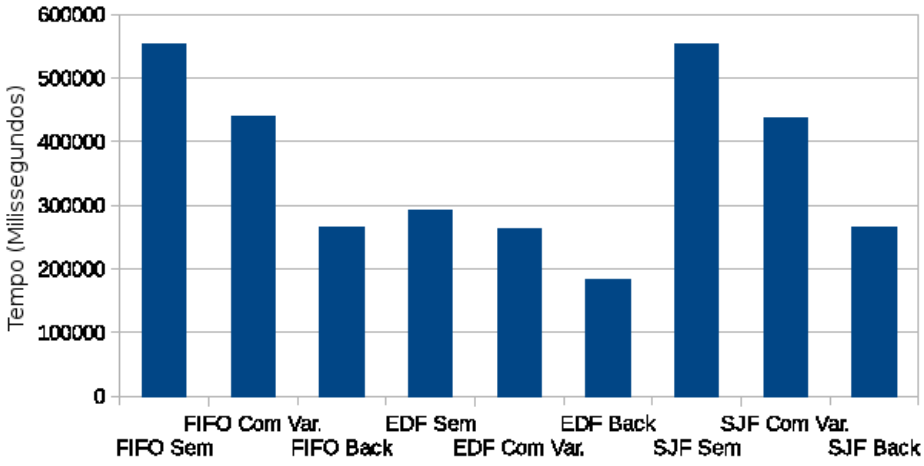


Figura 5. Média de *Turnaround* dos *jobs*.

Através dos resultados foi possível verificarmos que na maioria dos casos é mais

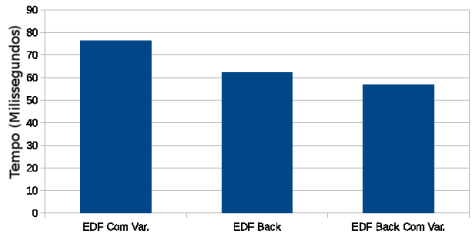
vantajosa a utilização do *Backfilling* em relação a variação para as métricas utilizadas. Isto é esperado uma vez que ignorou-se o tempo de chegada e com isso aumentou consideravelmente a probabilidade de achar um job (via *backfilling*) que poderia ser escalonado naquele momento. Porém a utilização do *Backfilling* desrespeita, de certa forma, a ordem de prioridade definida pela fila. Isto acontece pois, enquanto a variação trabalha com a simples "negociação" da demanda por recursos necessários para a execução do *job* que aguarda no inicio da fila, a técnica de *Backfilling* consiste em buscar pela ordem de prioridade da fila, qual o *job* que se encaixa na disponibilidade de recursos para aquele momento. A fim de minimizar o atraso gerado pela técnica de *Backfilling* ao *job* do inicio da fila, uma alternativa seria combinar as duas abordagens. Neste novo modelo, uma vez que não há recursos suficientes para atender ao *job* do inicio da fila, antes de iniciar o *Backfilling*, é realizada a negociação da variação de recursos com este *job*. Somente se, mesmo com a variação, não for possível atender ao *job* do inicio da fila, é que será realizado o *Backfilling*.

	Jobs Atrasados	Somatório Atrasos	Média Turn	Tempo Total
FIFO	279	117.519.984	5.541.84	1.373.646
FIFO com Var.	263	89.527.209	439.845	1.378.620
FIFO Backfilling	239	46.855.116	265.535	1.223.244
FIFO Backfilling com Var.	251	69.756.312	347.093	1.364.832
EDF	98	2.473.678	291.651	1.400.228
EDF com Var.	76	2.347.550	264.676	1.353.195
EDF Backfilling	62	17.585.390	183.177	1.436.243
EDF Backfilling com Var.	57	11.558.598	174.635	1.409.433
SJF	279	117.488.354	554.241	1.374.300
SJF com Var.	263	90.032.299	438.835	1.374.357
SJF Backfilling	238	46.789.658	265.225	1.227.557
SJF Backfilling com Var.	251	68.850.665	345.003	1.357.569

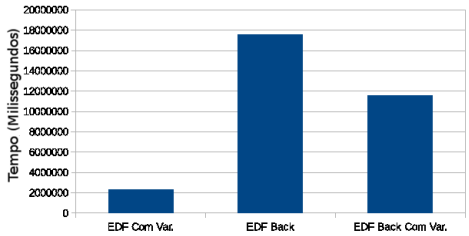
Tabela 2. Consolidação dos resultados obtidos para cada métrica em cada uma das abordagens.

Partindo deste principio, foi implementada e testada esta abordagem de *Backfilling* com variação para cada um dos tipos de filas testados e comparados com os demais. Com os resultados obtidos foi possível perceber que, para as filas FIFO e SJF, o desempenho foi pior que as suas abordagens com *Backfilling* e melhor que as suas abordagens somente com a variação. Porém a utilização desta abordagem se mostrou vantajosa com filas EDF para determinadas métricas, com destaque para a redução de 9% e 25% da quantidade de *jobs* atrasados em relação ao EDF com *Backfilling* e EDF com variação, respectivamente (Figura 6(a)). Em relação ao somatório dos atrasos, o seu desempenho ficou 34% melhor que o EDF com *Backfilling*, porém este tempo corresponde a quase 5 vezes o tempo para o EDF com variação apenas (Figura 6(b)). No que se refere a média de *turnaround*, o seu desempenho foi melhor que os demais, apresentando uma redução de 5% e 34% do tempo de *turnaround* em relação ao EDF com *Backfilling* e EDF com variação, respectivamente (Figura 6(c)). A Tabela 2 apresenta os valores numéricos de cada teste realizado neste experimento para as quatro métricas citadas anteriormente.

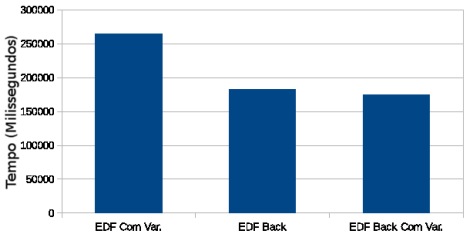
Neste experimento foi adicionada ainda uma nova métrica com um perfil um pouco diferente das demais. Esta métrica permite o acompanhamento do *turnaround* dos



(a) *Jobs* Atrasados



(b) Somatório dos atrasos



(c) Média de *Turnaround*

Figura 6. Resultados da utilização de *Backfilling* com a variação.

jobs em etapas de forma a avaliar o comportamento de cada abordagem não apenas ao final da execução mas também durante o processo de escalonamento. Esta métrica visa permitir uma análise do escalonamento com base em dados intermediários diferentemente das demais que representam dados finais. Desta forma, a cada 10 *jobs* terminados, foi calculada a média de *turnaround*, totalizando 40 *steps* até o final da execução. Os gráficos a seguir apresentam os resultados obtidos separados de acordo com o tipo de fila utilizada.

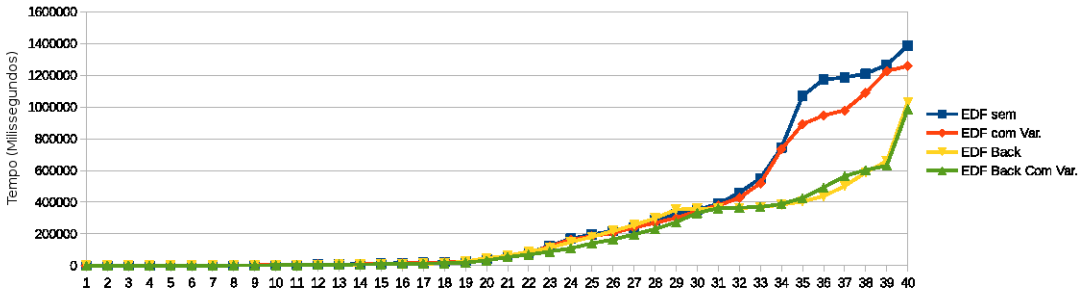


Figura 7. *Turnaround step by step* para a fila EDF.

O gráfico da Figura 7 apresenta os valores referentes a média de *turnaround* a cada conjunto de 10 *jobs* para as abordagens com filas EDF. Este tipo de fila reordena os *jobs* dando maior prioridade aos que possuem menor *deadline*. Como, neste trabalho, o *deadline* é baseado no tempo de execução, *jobs* menores possuem menor *deadline*. Desta forma, os maiores *jobs* e que consomem mais recursos ficam para o final. Um outro importante favor notado é que a variação do *turnaround* é pequena inicialmente e vai crescendo gradativamente. Já nos últimos *steps* acontecem variações maiores devido ao processamento desses *jobs* maiores. Podemos ver através do gráfico que a diferença entre

as abordagens se torna maior exatamente quando os *jobs* são maiores, *jobs* pequenos tem menor impacto, e a utilização do EDF com variação permite que a curva de crescimento do tempo seja mais leve em certos momentos em relação ao EDF básico. Em geral podemos perceber que o EDF com *Backfilling*, apesar de, em certos momentos, obter tempos piores que os demais, possui um dos melhores resultados, próximo do EDF com *Backfilling* e variação que se manteve mais constante como a melhor alternativa para este *workload*.

5. Conclusão

Com base nos testes realizados neste trabalho podemos concluir que a utilização da variação permite a melhora do desempenho em relação às abordagens básicas mantendo a ordem de prioridade estabelecida pela fila utilizada. Porém, de uma forma geral, a utilização de *Backfilling* mostrou-se mais vantajosa em relação as demais abordagens testadas. A possibilidade de utilizar uma técnica como complemento da outra não foi vantajosa para FIFO e SJF, porém mostrou-se promissora para ser testada com outros *workloads* com filas EDF. Outra importante análise a ser feita é referente a utilização de métricas que oferecem informações a cada etapa do escalonamento durante a execução, permitindo uma análise mais precisa do escalonamento e a identificação de possíveis gargalos na abordagem utilizada.

Neste trabalho limitamos os experimentos a utilização de apenas um modelo de variação que foi considerado como um limite superior aos experimentos realizados com o Linpack. Como trabalhos futuros pretende-se realizar novos experimentos com diferentes valores de variação para diferentes perfis de aplicações. Os resultados obtidos possibilitarão uma análise mais aprofundada com extrapolações para casos onde a aplicação tenha um comportamento melhor apesar da variação dos recursos.

Este trabalho tem como um dos seus principais objetivos nos seus trabalhos futuros a extrapolação para o escalonamento de aplicações adaptativas (maleáveis ou moldáveis) e o escalonamento destas em conjunto com aplicações rígidas.

Referências

Buyya, R. and Murshed, M. (2002). Gridsim: A toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing. *CONCURRENCY AND COMPUTATION: PRACTICE AND EXPERIENCE (CCPE*, 14(13):1175–1220.

Chen, B. and Matis, T. I. (2013). A flexible dispatching rule for minimizing tardiness in job shop scheduling. *International Journal of Production Economics*, 141(1):360 – 365. Meta-heuristics for manufacturing scheduling and logistics problems.

Cirne, W. and Berman, F. (2002). Using moldability to improve the performance of supercomputer jobs. *J. Parallel Distrib. Comput.*, 62(10):1571–1601.

Dutot, P., Netto, M. A. S., Goldman, A., and Kon, F. (2005). Scheduling moldable BSP tasks. In *Job Scheduling Strategies for Parallel Processing, 11th International Workshop, JSSPP 2005, Cambridge, MA, USA, June 19, 2005, Revised Selected Papers*, pages 157–172.

Feitelson, D. G., Rudolph, L., Schwiegelshohn, U., Sevcik, K. C., and Wong, P. (1997). Theory and practice in parallel job scheduling. In *Proceedings of the Job Scheduling*

Strategies for Parallel Processing, IPPS '97, pages 1–34, London, UK, UK. Springer-Verlag.

Huang, K.-C., Huang, T.-C., Tsaia, M.-J., Chang, H.-Y., and Tung, Y.-H. (2013a). Moldable job scheduling for HPC as a service with application speedup model and execution time information. *J. Convergence*, 4(4):14–22.

Huang, K.-C., Huang, T.-C., Tung, Y.-H., and Shih, P.-Z. (2013b). *Advances in Intelligent Systems and Applications - Volume 2: Proceedings of the International Computer Symposium ICS 2012 Held at Hualien, Taiwan, December 12–14, 2012*, chapter Effective Processor Allocation for Moldable Jobs with Application Speedup Model, pages 563–572. Springer Berlin Heidelberg, Berlin, Heidelberg.

Hunold, S. (2015). One step toward bridging the gap between theory and practice in moldable task scheduling with precedence constraints. *Concurrency and Computation: Practice and Experience*, 27(4):1010–1026.

Jansen, K. (2012). A $(3/2+\epsilon)$ approximation algorithm for scheduling moldable and non-moldable parallel tasks. In *Proceedings of the Twenty-fourth Annual ACM Symposium on Parallelism in Algorithms and Architectures*, SPAA '12, pages 224–235, New York, NY, USA. ACM.

Li, K., Zheng, W., Wu, Y., and Yuan, Y. (2014). Guarantee strict fairness and utilize prediction better in parallel job scheduling. 25(4):971–981.

Liu, F. and Weissman, J. B. (2015). Elastic job bundling: An adaptive resource request strategy for large-scale parallel applications. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '15, pages 33:1–33:12, New York, NY, USA. ACM.

Lublin, U. and Feitelson, D. G. (2003). The workload on parallel supercomputers: Modeling the characteristics of rigid jobs. *J. Parallel Distrib. Comput.*, 63(11):1105–1122.

Saule, E., Bozdağ, D., and Catalyurek, U. V. (2010). *Job Scheduling Strategies for Parallel Processing: 15th International Workshop, JSSPP 2010, Atlanta, GA, USA, April 23, 2010, Revised Selected Papers*, chapter A Moldable Online Scheduling Algorithm and Its Application to Parallel Short Sequence Mapping, pages 93–109. Springer Berlin Heidelberg, Berlin, Heidelberg.

Utrera, G., Tabik, S., Corbalan, J., and Labarta, J. (2012). *Euro-Par 2012 Parallel Processing: 18th International Conference, Euro-Par 2012, Rhodes Island, Greece, August 27-31, 2012. Proceedings*, chapter A Job Scheduling Approach for Multi-core Clusters Based on Virtual Malleability, pages 191–203. Springer Berlin Heidelberg, Berlin, Heidelberg.

Wu, S., Tuo, Q., Jin, H., Yan, C., and Weng, Q. (2015). Hrf: A resource allocation scheme for moldable jobs. In *Proceedings of the 12th ACM International Conference on Computing Frontiers*, CF '15, pages 17:1–17:8, New York, NY, USA. ACM.

Yang, H.-a., Sun, Q.-f., Saygin, C., and Sun, S.-d. (2012). Job shop scheduling based on earliness and tardiness penalties with due dates and deadlines: an enhanced genetic algorithm. *The International Journal of Advanced Manufacturing Technology*, 61(5-8):657–666.