Um Algoritmo de Escalonamento com Intercalação de Processos em Grades Computacionais

Luiz F. Bittencourt, Edmundo R. M. Madeira

[1]Instituto de Computação – Universidade Estadual de Campinas (UNICAMP)
Caixa Postal 6176 – 13084-971 – Campinas – SP – Brasil

{bit,edmundo}@ic.unicamp.br

Abstract. *Computational grids are nowadays being used as a collaborative environment to e-Science. Also, the workflow model is becoming the standard in e-Science to represent dependent tasks. Consequently, the development of algorithms to schedule workflows considering its dependencies and the grid characteristics are of major interest. In this paper we present an algorithm to schedule multiple workflows potentially sharing the same set of heterogeneous resources. The algorithm queues tasks by interleaving pieces of each submitted workflow. Experimental results show that by interleaving workflows, the gaps in the queue, generated by data transfers of dependencies, are used to process other tasks, minimizing the average workflows execution time.*

Resumo. *Grades computacionais estão sendo usadas atualmente como um ambiente colaborativo para a e-Ciência. Ainda, o modelo de* workflow *está se tornando o padrão na e-Ciência para a representação de tarefas dependentes. Então, o desenvolvimento de algoritmos para escalonamento de* workflows *considerando suas dependências e as características das grades são de grande interesse. Neste artigo apresentamos um algoritmo que, ao enfileirar tarefas intercalando pedaços de cada* workflow*, escalona múltiplos* workflows *no mesmo conjunto de recursos heterogêneos. Experimentos mostram que os espaços gerados pela transmissão de dados entre tarefas são utilizados para processar outras tarefas, minimizando o tempo médio de execução dos* workflows.

1. Introdução

Ambientes heterogêneos e compartilhados, como as grades computacionais, apresentam desafios adicionais quando tratamos de escalonamento. A heterogeneidade insere novas variáveis nas decisões do algoritmo, enquanto o compartilhamento de recursos insere imprevisibilidade e características probabilísticas. Além disso, nesse tipo de ambiente a concorrência entre processos está sempre presente, pois tarefas enviadas por diferentes indivíduos compartilham o mesmo conjunto de recursos computacionais.

Tarefas podem ser únicas e independentes ou podem ser dependentes, compondo um processo onde cada tarefa depende de dados computados por outras tarefas que a antecedem. Processos podem ser modelados como *workflows*, utilizando um grafo acíclico direcionado (*directed acyclic graph* - DAG) para representar as tarefas e suas dependências. O escalonamento de tarefas dependentes é um problema NP-Completo [El-Rewini et al. 1995], portanto o aprimoramento constante de algoritmos e

heurísticas é fundamental para melhorar o desempenho na execução de *workflows* em grades [Taylor et al. 2007]. As palavras *processo* e *workflow* usadas durante o texto referem-se ao DAG composto pelas tarefas dependentes.

Escalonamento, predição de desempenho e re-escalonamento estão recebendo atenção substancial de pesquisadores em grades [Chen and Maheswaran 2002, Prodan and Fahringer 2005, Sun and Wu 2003]. Entretanto, um aspecto importante não está recebendo a atenção merecida: o escalonamento e execução concorrente de processos com tarefas dependentes. Assim, neste trabalho exploramos as dependências e os intervalos de tempo gerados pela transmissão de dados para aumentar a utilização dos recursos e minimizar o tempo de execução dos processos.

No escalonamento de processos com tarefas dependentes surgem espaços entre as tarefas, que são provenientes do tempo de transmissão de dados entre tarefas que estão em recursos diferentes. Considerando o escalonamento de múltiplos processos, podemos, de forma geral, utilizar quatro estratégias:

- escalonar processos seqüencialmente: processos são escalonados em ordem de chegada. Tarefas só podem ser escalonadas após todas as tarefas dos processos já escalonados.
- preencher espaços presentes no escalonamento de processos anteriores: processos são escalonados em ordem de chegada. Tarefas podem ser escalonadas em espaços deixados por outros processos já escalonados desde que o tempo de execução de tais tarefas não ultrapasse o tamanho do espaço considerado, dando certa prioridade aos processos já escalonados. Resulta em algoritmos mais complexos, aumentando o número de escalonamentos possíveis.
- escalonar tarefas que ainda não foram executadas intercalando tarefas de processos distintos: quando um processo é escalonado, este tem suas tarefas intercaladas com tarefas dos processos já escalonados que ainda não foram enviadas para execução. Também permite priorizar processos, porém de forma mais maleável.
- escalonar tarefas considerando que poderão executar concomitantemente: processos são escalonados em recursos com menor número de tarefas na tentativa de minimizar uso de memória, processador e acessos ao disco. Tarefas de diferentes processos executam ao mesmo tempo, competindo por recursos.

A primeira estratégia proporciona algoritmos simples, porém não aproveita o tempo de transmissão de dados para processar tarefas de outros processos. A segunda, por sua vez, aproveita os espaços para processar outras tarefas sem interferir no escalonamento de processos escalonados anteriormente, dando certa prioridade a processos que chegaram antes. A terceira aproveita os espaços de transmissão de dados de um processo através da inserção de tarefas de outros processos já durante o processo inicial de escalonamento, o que pode alterar o tempo de execução de processos já escalonados. A quarta estratégia simplifica o algoritmo de escalonamento, porém torna muito difícil prever o tempo de execução dos processos, pois a competição por recursos como memória, disco e largura de banda pode atrasar a execução de forma não mensurável.

Neste trabalho apresentamos um novo algoritmo de escalonamento de tarefas dependentes que utiliza a terceira estratégia citada. Esse algoritmo permite priorizar processos e, ao mesmo tempo, flexibilizar essa prioridade de acordo com cada cenário. O algoritmo proposto intercala partes de *workflows*, aproveitando tempo de transmissão de dados

de um *workflow* para realizar processamento de tarefas de outros *workflows*, além de continuar permitindo previsões do tempo de execução de cada processo. A flexibilização da prioridade permite que processos que estão executando há muito tempo sejam priorizados de forma mais severa que processos recém iniciados, abrindo caminho para a introdução de um controle de *deadline* no algoritmo. Realizamos comparações do algoritmo apresentado com outras duas estratégias citadas, apresentando conclusões, embasadas por resultados experimentais, sobre os prós e contras de cada abordagem.

Na Seção 2 introduzimos o problema de escalonamento de tarefas dependentes, comentando alguns trabalhos relacionados a este tema na Seção 3. Uma visão geral dos algoritmos utilizados pelo escalonador é descrita na Seção 4. O algoritmo proposto é apresentado na Seção 5 e resultados experimentais são analisados na Seção 6. A Seção 7 finaliza este artigo com as conclusões e alguns comentários.

2. Escalonamento de Tarefas

O problema de escalonamento de tarefas consiste em, dado um conjunto de tarefas dependentes e suas dependências, escolher em qual recurso cada tarefa irá executar. Um escalonador, em geral, tem uma função objetivo. Por exemplo, um escalonador pode tentar minimizar o tempo de execução (*makespan*) do processo, maximizar a utilização de recursos ou maximizar o *throughput* do sistema. Neste trabalho focamos na minimização do tempo de execução de múltiplos processos para melhorar a utilização dos recursos e, conseqüentemente, reduzir o tempo de obtenção de resultados para processos executados na grade.

Um *workflow* é geralmente representado por um grafo acíclico direcionado (DAG) $G = (V, E)$, onde V é o conjunto de tarefas a serem executadas e E é o conjunto de arestas que representam as dependências entre tarefas. Portanto, uma aresta direcionada $e_{i,j}$, com sua origem na tarefa t_i e destino na tarefa t_j, significa que t_j pode iniciar sua execução somente após t_i finalizar e enviar todos os dados necessários à t_j.

Sendo o escalonamento de tarefas um problema NP-Completo, o problema de otimização associado é NP-Difícil. Portanto, desenvolvemos uma heurística e uma abordagem adaptativa para escalonar tarefas em grades [Bittencourt and Madeira 2007b], onde utilizamos um esquema de execução em quatro passos, como apresentado na Figura 1. No primeiro passo é utilizado o algoritmo *Path Clustering Heuristic* (PCH), que realiza o escalonamento inicial dos processos. A técnica de execução baseada em *rounds*, apresentada em [Bittencourt and Madeira 2007b], é utilizada no segundo passo, onde o algoritmo decide dinamicamente em cada *round* quais tarefas escalonadas serão enviadas para execução. Com isso, o algoritmo obtém informações do desempenho de cada recurso para tomar decisões de re-escalonamento. O módulo dinâmico decide quais tarefas serão executadas em cada *round*, amparado pelo módulo adaptativo, que regula o tamanho de cada *round* em cada recurso. Essa regulagem adaptativa do tamanho dos *rounds* tenta evitar o envio de muitas tarefas para execução em recursos que tiveram alta variação de desempenho no passado. O tamanho dos *rounds* é relativo ao tempo de execução das tarefas em cada recurso, dependendo da capacidade do recurso em questão. A execução é o terceiro passo, e o re-escalonamento o quarto. Introduzimos brevemente os algoritmos referentes a cada passo na Seção 4.

Neste trabalho apresentamos um algoritmo dinâmico para escalonamento de

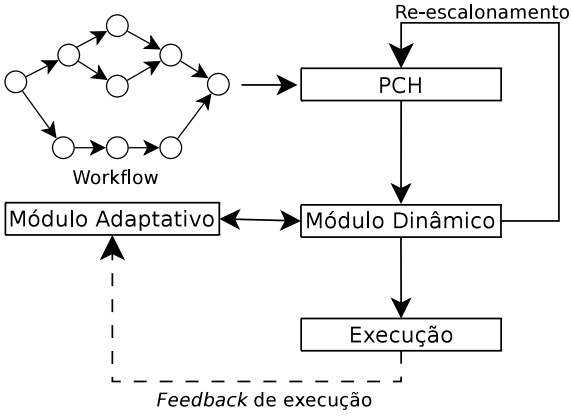

Figura 1. Esquema de execução em quatro passos.

múltiplos processos compostos de tarefas dependentes. Uma modificação na heurística PCH foi feita para obter melhor ajuste ao intercalar processos e evitar que espaços gerados por transmissão de dados permaneçam no escalonamento. O objetivo principal do algoritmo é minimizar o tempo de execução dos processos que são submetidos para execução na grade.

3. Trabalhos Relacionados

Existem trabalhos que exploram escalonamento de tarefas tanto em ambientes homogêneos ([Kwok and Ahmad 1996]) como em ambientes heterogêneos ([Sakellariou and Zhao 2004, Topcuoglu et al. 2002]).

Dois conhecidos algoritmos de escalonamento de tarefas para sistemas heterogêneos são o *Heterogeneous Earliest Finish Time* (HEFT) [Topcuoglu et al. 2002] e o *Critical Path on a Processor* (CPOP) [Topcuoglu et al. 2002]. Esses algoritmos escalonam tarefas dependentes em recursos heterogêneos considerando as diferenças de capacidade de processamento e largura de banda dos recursos, porém não consideram variação de desempenho e comportamento dinâmico do ambiente.

Um meta-escalonador de DAGs para grades é o DAGMan [Frey 2002]. Sua proposta é realizar uma pré-seleção de tarefas prontas para serem executadas e, de acordo com essa seleção, enviar as tarefas de forma independente ao escalonador. Então, o escalonador realiza o procedimento de escalonamento das tarefas sem considerar as dependências entre elas, o que pode acarretar altos custos de comunicação.

Em [Prodan and Fahringer 2005], um estudo de caso em escalonamento dinâmico para *workflows* científicos em grades é apresentado. É proposto um re-escalonamento dinâmico com iterações sobre o *workflow*, gerando um DAG com tarefas que devem ser re-escalonadas em cada iteração. Após isso, o DAG gerado é escalonado, conseqüentemente realizando um re-escalonamento de suas tarefas.

Um escalonador dinâmico em dois níveis para *wide area networks* (WANs) é proposto em [Chen and Maheswaran 2002], onde um escalonador de nível macro consulta escalonadores no segundo nível para selecionar em que LAN o processo será executado. Não há mecanismo para dividir o processo para execução em mais de uma LAN, o que pode ocasionar tempos de execução maiores. Ainda, todos os escalonadores no se-

gundo nível devem responder a todas as requisições de escalonamento. Se houver muitas requisições pode haver sobrecarga nos escalonadores.

Apesar de existirem vários trabalhos que tratam da execução de *workflows* em grades [Yu and Buyya 2005], não temos conhecimento de trabalhos que tratam da execução de vários processos ao mesmo tempo na grade, levando em consideração perda de desempenho, o ambiente dinâmico e re-escalonamento. Assim, consideramos que este é um problema em aberto no qual focamos este artigo.

4. Visão Geral dos Algoritmos

Nesta seção apresentamos uma visão geral dos algoritmos utilizados como base para o algoritmo de intercalação, além de referências com maiores detalhes. Como na maioria dos trabalhos de escalonamento, os algoritmos assumem que o middleware e o modelo de programação fornecem os mecanismos de detecção de término de tarefas e saída de recursos, além de informações sobre custos de computação e comunicação das tarefas.

4.1. Path Clustering Heuristic

O algoritmo Path Clustering Heuristic (PCH) utiliza a técnica de *clustering* para criar grupos (clusters) de tarefas. Agrupando caminhos do DAG o PCH reduz os custos de comunicação entre tarefas, pois todas as tarefas de um mesmo *cluster* são escalonadas no mesmo recurso. O algoritmo utiliza alguns atributos calculados para cada tarefa, estimando os tempos de início (EST - *Earliest Start Time*) e término (EFT - *Estimated Finish Time*) das tarefas e do processo. Esses atributos são calculados utilizando informações fornecidas pelo middleware e pelo modelo de programação. Detalhes podem ser encontrados em [Bittencourt and Madeira 2007b]. Neste trabalho utilizamos uma versão modificada do algoritmo PCH no algoritmo de intercalação. O termo PCH de agora em diante refere-se ao algoritmo modificado.

Para criar os *clusters* de tarefas, o PCH realiza uma busca em profundidade no DAG, sempre selecionando a tarefa não escalonada que está no maior caminho que inicia na primeira tarefa e termina na última tarefa do DAG. O algoritmo pára a busca quando encontra uma tarefa que tem algum predecessor não escalonado, e esta tarefa não é incluída no *cluster*. O *cluster* é escalonado no recurso que proporciona menor tempo estimado de término para sua última tarefa, e então o próximo *cluster* é formado utilizando a mesma busca em profundidade. A Figura 2 mostra um exemplo simples de *clusters* formados pelo PCH e do escalonamento resultante em dois recursos de mesmo poder de processamento. Primeiramente o *cluster* 0 é formado, onde a busca em profundidade inicia-se na tarefa 1, seguindo pelas tarefas 2, 3, 4, 7, e, ao encontrar a tarefa 9, a busca cessa e esta tarefa não é adicionada ao *cluster*. Esse *cluster* é, então, escalonado no recurso 0. O *cluster* 1 é composto pelas tarefas 6 e 8, pois essas tarefas estão no segundo maior caminho entre a primeira e a última tarefa do processo. Após o *cluster* 1 ser escalonado no recurso 1, que lhe proporciona menor tempo estimado de término que o recurso 0, o *cluster* 2 é formado pelas tarefas 5, 9 e 10 e escalonado no recurso 0.

4.2. PCH Dinâmico

Após realizado o escalonamento inicial, as tarefas devem ser enviadas aos respectivos recursos para que sejam executadas. Em um ambiente onde os recursos são compartilhados e pode haver variação no desempenho, enviar as tarefas tomando como certa a

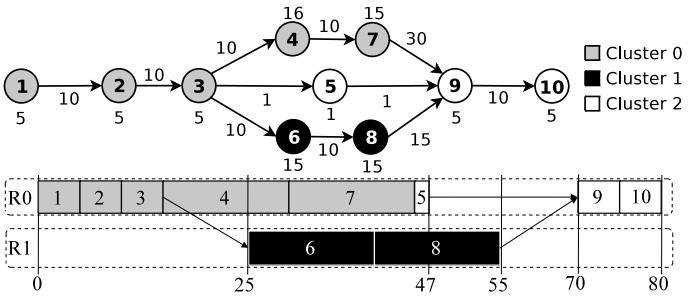

Figura 2. Exemplo de escalonamento utilizando o PCH.

execução nos recursos atribuídos no escalonamento inicial pode atrasar o recebimento dos resultados. Se outros processos independentes da grade são iniciados nos recursos, o escalonamento inicialmente planejado pode tornar-se muito mais longo que a estimativa. Assim, um algoritmo dinâmico é necessário nesse tipo de ambiente.

Utilizando o escalonamento provido pelo PCH, as tarefas são enviadas para execução considerando o conceito de *rounds* apresentado em [Bittencourt and Madeira 2007b]. Em cada *round* de execução, o PCH dinâmico seleciona um conjunto de tarefas que serão enviadas para execução. A seleção de parte do DAG para execução é feita da seguinte maneira: uma tarefa n é enviada para execução no *round* número k (de um total de T *rounds*) se n não iniciou sua execução e se $EFT_n \leq \frac{makespan}{T/k}$, onde EFT_n é o tempo estimado de término da tarefa n e $1 \leq k \leq T$, ou se n é a primeira tarefa no recurso onde está escalonada. À medida que as tarefas terminam de executar, o escalonador pode comparar o tempo de execução real com o tempo estimado nos atributos de cada tarefa. Se o desempenho de um recurso está aquém do desempenho esperado, respeitando-se um dado limiar ou margem de erro (por exemplo, 10%), as tarefas que ainda não foram enviadas para execução em tal recurso são re-escalonadas utilizando o algoritmo PCH. Os *rounds* de execução são repetidos até o término do processo.

4.3. Extensão Adaptativa

A extensão adaptativa tem como objetivo ajustar o tamanho dos *rounds* de execução de acordo com certas características do grafo e dos recursos. Ao selecionar quantas tarefas serão enviadas em determinado *round*, a extensão adaptativa leva em conta o tamanho do grafo, os pesos das tarefas e o desempenho dos recursos na execução das tarefas anteriores, tanto do mesmo processo quanto de processos já finalizados. Mais detalhes podem ser encontrados em [Bittencourt and Madeira 2007b].

Um modelo matemático foi desenvolvido para que as características citadas acima sejam consideradas ao definir o tamanho dos *rounds*. Com isso, o módulo dinâmico é agora amparado pelo modo adaptativo, que fornece informações para adaptar o tamanho dos *rounds* de acordo com o *feedback* de execução. O tamanho dos *rounds* é independente em cada recurso. O tamanho do *round* inicial é dependente do tamanho das tarefas a serem executadas e os *rounds* subseqüentes são adaptados de acordo com o desempenho dos recursos. Com isso, tentamos evitar enviar muitas tarefas para executar em recursos que têm histórico de alta variação no desempenho através da redução do tamanho dos *rounds* em tais recursos. Por outro lado, recursos que na maioria das vezes têm o desem-

penho esperado (ou próximo do esperado) recebem mais tarefas por *round*. Através desse mecanismo tentamos evitar o re-escalonamento de tarefas.

4.4. Busca de Espaços

Quando escalonamos um processo com tarefas dependentes em recursos distribuídos, o tempo de transmissão de dados proveniente das dependências resulta em espaços ociosos nos recursos. Um exemplo desses espaços pode ser visto entre as tarefas 5 e 9 na Figura 2. Dessa forma, se temos processos sendo executados e outros processos são submetidos à grade, podemos utilizar esses espaços ao escalonar novos processos.

Apresentamos em [Bittencourt and Madeira 2007a] um algoritmo para realizar preenchimento desses espaços sem interferir nos processos já escalonados. Em grades computacionais a variação de desempenho dos recursos dificulta essa tarefa, pois um atraso na execução de tarefas que foram escalonadas em tais espaços irá atrasar também a execução do processo que já estava escalonado no recurso. Para contornar esse problema, o algoritmo desenvolvido utiliza margens de segurança ao escolher em que espaços um dado *cluster* pode ser escalonado. Com isso, os espaços ociosos são usados para execução de outros processos, minimizando o tempo de execução global dos processos, com mínima interferência no tempo que foi inicialmente estimado para execução de cada processo.

5. O Algoritmo de Intercalação

Nesta seção propomos um novo algoritmo que intercala processos com o objetivo de minimizar o tempo de execução dos *workflows*. A intercalação de processos pode ser abordada de forma a priorizar ou não processos que já estão escalonados. Essa priorização pode ser feita considerando características dos processos, tais como pesos das tarefas e custos de comunicação, tamanho do caminho crítico dos processos, quantidade de *clusters* de tarefas de cada processo, e assim por diante.

No cenário de escalonamento de múltiplos processos, à medida que processos chegam para serem escalonados, o passo de seleção de tarefas deve considerar as tarefas não executadas de processos já escalonados e dos processos que chegam. Para realizar essa seleção nosso algoritmo considera que o primeiro processo a chegar, esteja ele já escalonado ou não, será o primeiro processo que terá tarefas selecionadas para escalonamento. Em seguida, o segundo processo terá tarefas selecionadas para escalonamento. Em suma, o algoritmo seleciona um grupo de tarefas de cada processo em ordem de chegada, escalonando cada grupo de tarefas selecionado, até que acabem os processos e tarefas a serem escalonadas. Durante o escalonamento de *clusters* intercalados, o algoritmo também realiza o preenchimento de espaços, buscando o melhor uso dos recursos disponíveis. Com essa intercalação de processos os espaços provenientes das transmissões de dados são utilizados para processamento de outros processos. O Algoritmo 1, que é executado no evento da chegada de um DAG G para escalonamento, mostra em mais detalhes este procedimento.

Na primeira linha, o Algoritmo 1 coleta os *clusters* que ainda não foram enviados para execução, enquanto na linha 2 cria o grupo *DAGs*, que contém os processos que contêm tais *clusters*. O processo que chegou para ser escalonado é adicionado ao grupo *DAGs* na linha 3. Na linha 4 uma política de prioridade pré-definida atribui a cada

Algoritmo 1 Visão geral da intercalação

1: $CLS_{fila} \Leftarrow$ Clusters não executados de processos escalonados.
2: $DAGs \Leftarrow$ Processos que têm tarefas em CLS_{fila}.
3: $DAGs \Leftarrow \{DAGs\} \cup \{G\}$.
4: Atribui prioridades a cada grafo de $DAGs$ de acordo com a política de prioridades.
5: **while** existem tarefas não escalonadas **do**
6: $G \Leftarrow$ próximo grafo com maior prioridade.
7: Escalona N clusters de G com procura de espaços, sendo N de acordo com a política de prioridades.
8: **end while**
9: Continua execução da fila de tarefas utilizando o algoritmo adaptativo dinâmico

processo um nível de prioridade que será usado para selecionar os processos e decidir quantos *clusters* de cada processo serão escalonados. A iteração entre as linhas 5 e 8 realiza o escalonamento dos processos, selecionando na linha 6 os grafos em seqüência de prioridade e na linha 7 realizando o escalonamento dos *clusters*, colocando cada *cluster* no recurso que proporciona menor EST para o sucessor de sua última tarefa.

A política de prioridades altera a ordem de escalonamento dos processos e o número de *clusters* de cada processo a serem escalonados em cada iteração do algoritmo. O valor N, utilizado na linha 7, representa o número de *clusters* do processo selecionado a serem escalonados. Esse valor é definido pela política de prioridades, com cada grafo tendo seu próprio N. Ainda, o valor de N pode ser variável para um mesmo grafo. Por exemplo, um grafo que já executou grande parte de suas tarefas pode ter sua prioridade aumentada pela política de prioridades. Ainda, a política de prioridades pode definir se um processo já escalonado pode ser re-escalonado na chegada de novos processos, ou a quantidade máxima de processos que podem ser re-escalonados no evento da chegada de um novo processo.

Nos experimentos realizados neste trabalho utilizamos prioridade FCFS (*First Come, First Served*), com $N = 1$ independente da prioridade atribuída a cada grafo. Assim, o escalonamento dos *clusters* é realizado de forma circular, começando no primeiro grafo que chegou e terminando quando não há mais *clusters* não escalonados. Note que, dessa forma, quanto maior um cluster, maior é o número de tarefas do grafo ao qual este *cluster* pertence que serão escalonadas. Dessa forma, a comunicação de tais tarefas será suprimida, enquanto *clusters* curtos serão intercalados com *clusters* de outros processos, tendo seu tempo de comunicação utilizado para processamento.

A Figura 3 mostra um exemplo do resultado do escalonamento de três processos com o algoritmo de intercalação, onde $P_0 = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$, $P_1 = \{A, B, C, D, E, F\}$, e $P_2 = \{a, b, c, d, e, f\}$, com P_2 sendo outra instância do processo P_1. Para tornar o exemplo simples, assumimos que os dois recursos possuem a mesma capacidade de processamento. O primeiro cluster a ser escalonado é o cluster $cls_{1,P0} = \{1, 2, 3, 4, 7\}$. Em seguida, os clusters $cls_{1,P1} = \{A, B, D\}$ e $cls_{1,P2} = \{a, b, d\}$ são escalonados. Então, retornando ao primeiro processo, o cluster $cls_{2,P0} = \{6, 8\}$ é escalonado. Continuando a iteração entre os processos, os clusters $cls_{2,P1} = \{C, E, F\}$ e $cls_{2,P2} = \{c, e, f\}$ são escalonados. Finalizando o processo, o cluster $cls_{3,P0} = \{5, 9, 10\}$ é escalonado. Note que há apenas um espaço entre as tarefas dos três processos, tornando

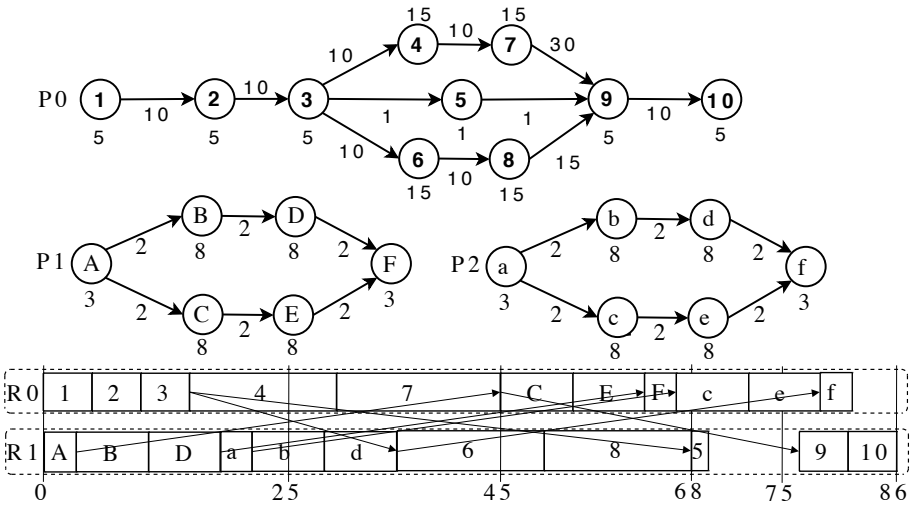

Figura 3. Exemplo de escalonamento utilizando intercalação.

baixa a taxa de ociosidade dos recursos.

Enquanto o algoritmo de preenchimento de espaço não interfere no escalonamento de processos já escalonados, a intercalação permite mecanismos de priorização de processos de acordo com políticas específicas. Por exemplo, um processo P_h que tem dependências de dados muito custosas pode ter um grupo escalonado primeiro, e outros processos com menor comunicação podem ser intercalados visando a utilização do processamento que estaria ocioso entre as transmissões de dados do processo P_h.

6. Resultados Experimentais

Nesta seção comparamos o algoritmo de intercalação com o algoritmo de escalonamento seqüencial (sem busca de espaços) e com o algoritmo de busca e preenchimento de espaços. A comparação foi feita escalonando três processos. Consideramos que grades são formadas por grupos de recursos. Por exemplo, um *cluster* de computadores ou uma LAN podem ser um grupo. A capacidade de comunicação entre dois recursos em um mesmo grupo é homogênea, porém diferente em grupos distintos. As conexões entre grupos são heterogêneas.

Utilizamos 15 grafos gerados aleatoriamente, com quantidade de nós entre 7 e 82. Comunicação baixa significa que todos os custos de comunicação em um dado processo são menores que todos os custos de computação. Comunicação alta significa que todos os custos de comunicação são maiores que todos os custos de computação. Comunicação média significa que todos os custos de computação e de comunicação foram gerados aleatoriamente dentro do mesmo intervalo. Para cada quantidade de grupos, variando de 2 a 25, três grafos foram escalonados 2500 vezes, com cada grupo tendo uma quantidade aleatória de recursos variando entre 1 e 5. Para cada execução, três processos foram selecionados aleatoriamente entre os 15 disponíveis, o que pode gerar 15^3 configurações diferentes de simulação.

As principais métricas utilizadas na literatura para comparação de escalonamento de tarefas são o Schedule Length Ratio (SLR) e o *speedup*. O SLR mostra quantas vezes o tamanho do escalonamento (*makespan*) é maior que a execução do caminho crítico do

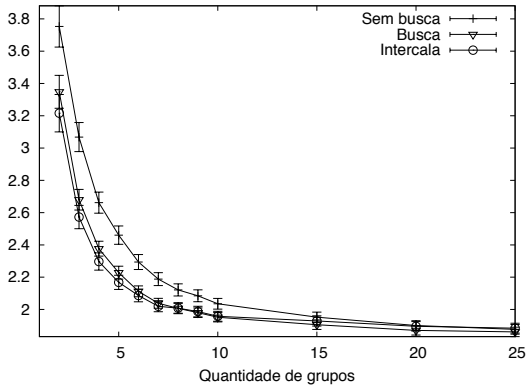

Figura 4. SLR médio para comunicação alta.

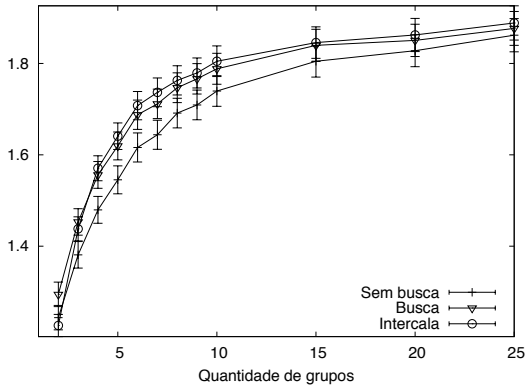

Figura 5. *Speedup* médio para comunicação alta.

DAG no melhor recurso disponível (quanto menor, melhor). O *speedup* mostra quantas vezes o *makespan* é menor que a execução de todas as tarefas em seqüência no melhor recurso disponível (quanto maior, melhor). Nas Figuras 4 a 9 são mostrados intervalos de confiança de 95%.

A Figura 4 mostra o SLR médio dos três processos para comunicação alta. Com 2 grupos, o ganho do algoritmo de busca de espaços em relação ao algoritmo sem busca de espaços é de cerca de 16%. Quando aumentamos o número de grupos esse ganho diminui, pois quanto mais grupos, mais recursos temos para escolher. Dessa forma, recursos sem tarefas escalonadas são utilizados em detrimento da utilização de espaços encontrados no escalonamento em outros recursos. Quando utilizamos o algoritmo de intercalação, há uma melhora em relação à utilização apenas da busca de espaços. O *speedup* para comunicação alta é mostrado na Figura 5. Notamos que quando há apenas 2 grupos de recursos, a intercalação de processos gera resultados equivalentes àqueles gerados pelo algoritmo sem busca de espaços, enquanto o algoritmo com busca de espaços gera resultados melhores. Porém, para 3 grupos o algoritmo de intercalação gera resultados equivalentes ao algoritmo de busca de espaços, sendo ambos melhores que o algoritmo sem busca de espaços. Isso se deve ao fato de que, tendo poucos recursos, há menos espaços a serem preenchidos, pois tarefas no mesmo recurso têm custo de comunicação igual a zero. Assim, os poucos espaços gerados são preenchidos pelo algoritmo de busca de espaços, enquanto o algoritmo de intercalação acaba gerando escalonamentos com *makespan* próximo àqueles gerados pelo algoritmo sem a busca de espaços, não havendo ganho na utilização de tempo de comunicação para processamento de tarefas de outros processos. Considerando entre 5 e 15 grupos, o algoritmo de intercalação é um pouco melhor que os outros dois algoritmos, conseguindo aproveitar mais tempo de comunicação para executar tarefas de outros processos do que o algoritmo que apenas busca espaços.

Os gráficos de SLR e *speedup* para comunicação média são mostrados nas Figuras 6 e 7, respectivamente. As curvas sem busca e com busca têm distância e comportamento semelhantes aos observados no gráfico para comunicação alta. A curva do algoritmo de intercalação neste cenário distanciou-se da curva do algoritmo com busca de espaços, pois ao termos tarefas e espaços de comunicação gerados aleatoriamente no mesmo intervalo, o algoritmo de busca de espaços encontra menos espaços onde pode encaixar tarefas quando

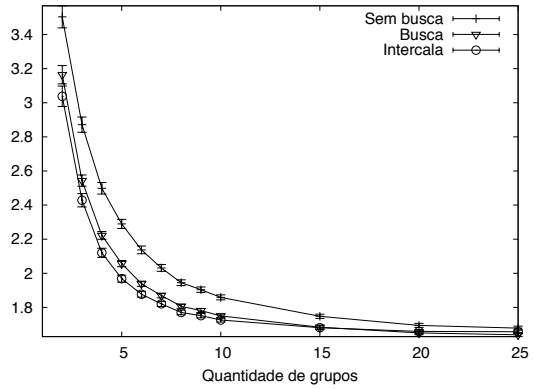

Figura 6. SLR médio para comunicação média.

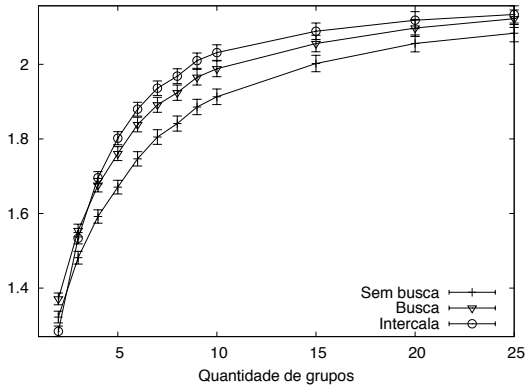

Figura 7. *Speedup* médio para comunicação média.

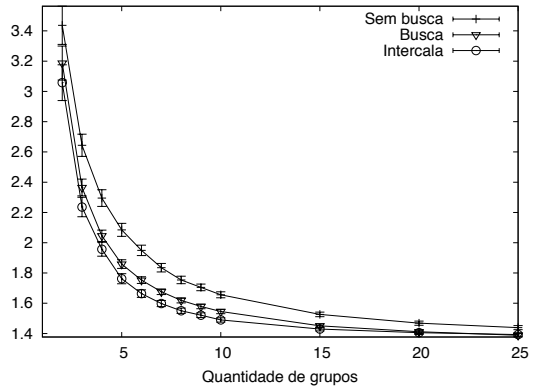

Figura 8. SLR médio para comunicação baixa.

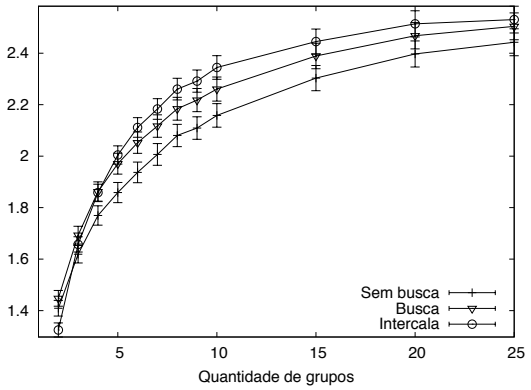

Figura 9. *Speedup* médio para comunicação baixa.

comparado à comunicação alta. Assim, o algoritmo de intercalação consegue melhores resultados, utilizando de forma mais eficaz o tempo de comunicação para processamento de tarefas de outros processos.

No cenário de comunicação baixa, os resultados de SLR (Figura 8) e *speedup* (Figura 9) mostram uma melhora do algoritmo de intercalação quando comparado ao algoritmo de busca de espaços. A mesma justificativa do cenário de comunicação média é válida: como os tempos de transmissão de dados são menores, sobram menos espaços para encaixar tarefas, tornando o algoritmo de busca de espaços menos eficaz. Por outro lado, o algoritmo de intercalação continua conseguindo utilizar os tempos de comunicação para computar tarefas de outros processos, minimizando as médias de SLR e *speedup*.

Após realizar comparações do escalonamento inicial resultante de cada algoritmo, realizamos simulações de execução dos três processos em um conjunto de recursos compartilhados. Dessa forma, podemos analisar como os escalonamentos iniciais de cada algoritmo se comportam em um ambiente dinâmico.

Sem re-escalonamento significa que todas as tarefas foram executadas nos recursos dados pelo escalonamento inicial, não importando seu desempenho, enquanto com re-escalonamento (representado por *re-esc.* nas figuras) significa que tarefas foram re-

escalonadas quando o recurso não obteve o desempenho esperado. O algoritmo de re-escalonamento utilizado é simples: caso o recurso não atinja o desempenho esperado, para cada cluster nesse recurso, re-escalone-o em outro recurso ordenando pelo menor tempo de início (EST) todas as tarefas desse recurso. Para determinar o desempenho dos recursos, primeiramente geramos um padrão de desempenho dos recursos em nosso laboratório, baseado em medições feitas nos computadores. No início de cada simulação, geramos o poder de processamento em cada intervalo de tempo para cada recurso, sendo as capacidades de processamento as mesmas para a execução dos três algoritmos.

As Figuras 10 e 11 mostram os resultados da simulação de execução com comunicação média. Podemos notar um comportamento semelhante ao encontrado nos escalonamentos iniciais: o algoritmo de intercalação consegue melhorar a média de SLR e speedup. As curvas do gráfico se comportam da mesma maneira que aquelas apresentadas no escalonamento inicial, sugerindo que a variação de desempenho dos recursos não afeta o ganho obtido pelo algoritmo de intercalação no escalonamento inicial.

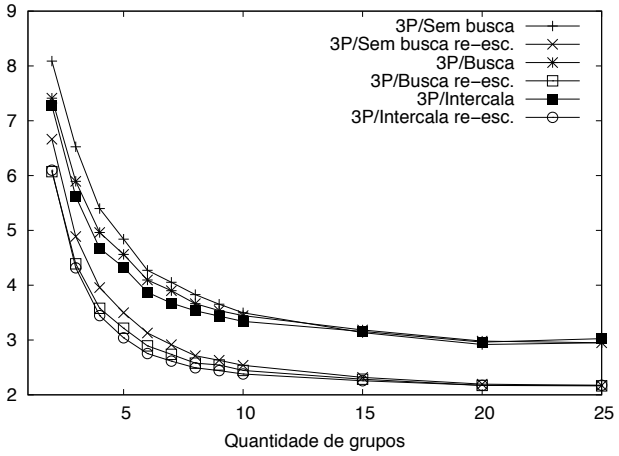

Figura 10. SLR médio para execuções com comunicação média.

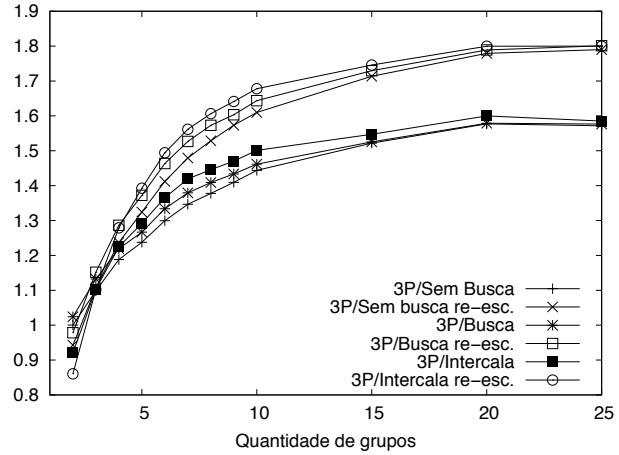

Figura 11. *Speedup* médio para execuções com comunicação média.

As Figuras 12 e 13, que mostram resultados das execuções com comunicação baixa, também seguem o padrão apresentado pelas figuras que mostram resultados do

escalonamento inicial, reforçando o resultado obtido para comunicação média. Com as

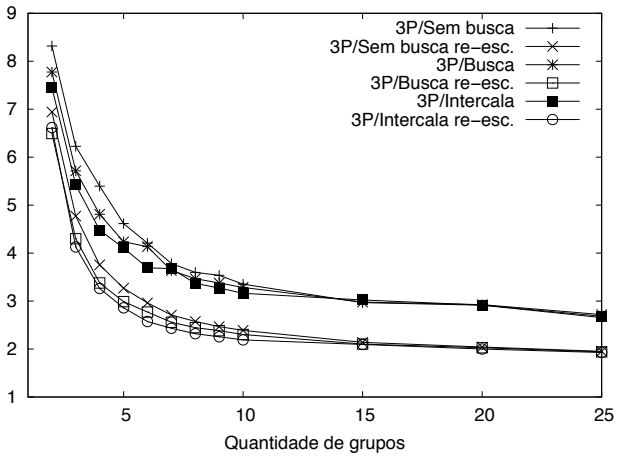

Figura 12. SLR médio para execuções com comunicação baixa.

simulações apresentadas tentamos comparar os três algoritmos tanto no escalonamento inicial como na execução em ambiente compartilhado. Os resultados sugerem que o algoritmo de intercalação melhora os resultados obtidos com o algoritmo de preenchimento de espaços, principalmente quando a comunicação entre tarefas é baixa.

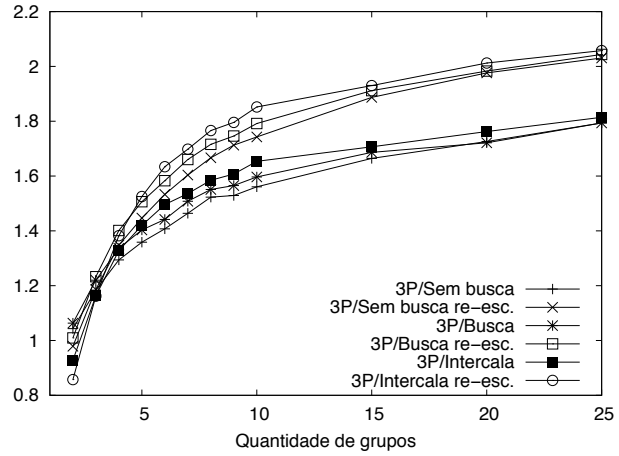

Figura 13. *Speedup* médio para execuções com comunicação baixa.

7. Conclusão

Quando algoritmos de escalonamento em grades são apresentados, até onde conhecemos, não é levado em consideração o escalonamento de múltiplos processos no mesmo grupo de recursos. Assumindo que as grades computacionais são compartilhadas e que o processamento de grandes quantidades de dados combina com as necessidades da e-Ciência, neste artigo apresentamos um algoritmo de escalonamento que considera múltiplos processos compartilhando o mesmo conjunto de recursos. Experimentos comparando o algoritmo com outros dois algoritmos, um de preenchimento de espaços e outro de escalonamento seqüencial, sugerem que o algoritmo de intercalação gera melhores escalonamentos que os outros dois algoritmos, principalmente quando os custos de comunicação entre tarefas são baixos.

Como trabalhos futuros podemos apontar o desenvolvimento de algoritmos de re-escalonamento avançados, a predição de desempenho dos recursos e a inclusão de um mecanismo que efetue medições e considere variação de desempenho na rede.

Agradecimentos

Os autores agradecem à FAPESP (05/59706-3) e ao CNPq (302307/2004-4) pelo apoio financeiro.

Referências

Bittencourt, L. F. and Madeira, E. R. M. (2007a). Fulfilling task dependence gaps for workflow scheduling on grids. In *3rd IEEE International Conference on Signal-Image Technology and Internet Based Systems*, Shanghai, China.

Bittencourt, L. F. and Madeira, E. R. M. (2007b). A performance oriented adaptive scheduler for dependent tasks on grids. *Concurrency and Computation: Practice and Experience, Online (In Press)*.

Chen, H. and Maheswaran, M. (2002). Distributed dynamic scheduling of composite tasks on grid computing systems. In *11th IEEE Heterogeneous Computing Workshop*, pages 119–128, Washington, DC, EUA. IEEE Computer Society.

El-Rewini, H., Ali, H. H., and Lewis, T. G. (1995). Task scheduling in multiprocessing systems. *IEEE Computer*, 28(12):27–37.

Frey, J. (2002). Condor DAGMan: Handling inter-job dependencies. http://www.cs.wisc.edu/condor/dagman/.

Kwok, Y.-K. and Ahmad, I. (1996). Dynamic critical-path scheduling: An effective technique for allocating task graphs to multiprocessors. *IEEE Transactions on Parallel and Distributed Systems*, 7(5):506–521.

Prodan, R. and Fahringer, T. (2005). Dynamic scheduling of scientific workflow applications on the grid: a case study. In *SAC '05: Proceedings of the 2005 ACM symposium on Applied computing*, pages 687–694, Santa Fe, EUA. ACM Press.

Sakellariou, R. and Zhao, H. (2004). A hybrid heuristic for dag scheduling on heterogeneous systems. In *13th Heterogeneous Computing Workshop*, pages 111,123, Santa Fe, EUA. IEEE Computer Society.

Sun, X.-H. and Wu, M. (2003). Grid harvest service: A system for long-term, application-level task scheduling. In *Proceedings of the 17th International Symposium on Parallel and Distributed Processing*, page 25.1, Nice, França. IEEE Computer Society.

Taylor, I. J., Deelman, E., Gannon, D. B., and Shields, M., editors (2007). *Workflows for e-Science. Scientific Workflows for Grids*. Springer Verlag.

Topcuoglu, H., Hariri, S., and Wu, M.-Y. (2002). Performance-effective and low-complexity task scheduling for heterogeneous computing. *IEEE Transactions on Parallel and Distributed Systems*, 13(3):260–274.

Yu, J. and Buyya, R. (2005). A taxonomy of scientific workflow systems for grid computing. *SIGMOD Records*, 34(3):44–49.