Process Migration: Controlling Application and Resource Dynamics by Combining Computation, Communication and Memory Metrics

Lucas Graebin, Rodrigo da Rosa Righi
Programa Interdisciplinar de Pós-Graduação em Computação Aplicada - Unisinos
Email: lgraebin@acm.org, rrrighi@unisinos.br

Philippe Olivier Alexandre Navaux
Programa de Pós-Graduação em Computação - Instituto de Informática - UFRGS
Email: navaux@inf.ufrgs.br

Abstract

In this paper we present MigBSP, a rescheduling model that acts on Bulk Synchronous Parallel applications. It combines the metrics Computation, Communication and Memory to make migration decisions on Computation Grids. MigBSP also offers efficient adaptations to reduce its own overhead. Additionally, MigBSP is infrastructure and application independent and tries to handle dynamicity on both levels. MigBSP's results show performance gains of up to 16% on dynamic environments while maintaining a small overhead when migrations do not take place.

1 Introdução

Ambientes de *Grids* Computacionais, ou somente, *Grids*, são intrinsecamente dinâmicos. Processadores podem ser compartilhados entre vários usuários, enquanto a rede pode se tornar congestionada durante alguns períodos do dia. Em nível de aplicação, processos podem mudar a sua quantidade de computação e/ou seu padrão de comunicação com os demais a qualquer momento. Consequentemente, o reescalonamento emerge como uma alternativa para remapear processos de maneira eficiente em *Grids*. Caso contrário, os benefícios de usar múltiplos processadores pode ser nulo[13]. Basicamente, as principais questões no tratamento do reescalonamento são: (i) como ele é oferecido para os desenvolvedores e; (ii) quais métricas serão usadas para compor a noção de carga.

A técnica de reescalonamento é principalmente oferecida pela inclusão de diretivas de migração ou através da implementação de balanceamento de carga iniciado pelo receptor[6]. Embora sejam largamente utilizadas, ambas as técnicas requerem um esforço no sentido de reconhecer a arquitetura paralela previamente e/ou mudar o código da aplicação. Além disso, uma abordagem comum de reescalonamento consiste em reorganizar os processos através da transferência periódica daqueles mais sobrecarregados para processadores mais rápidos. Além da perspectiva de computação, reescalonadores cientes de métricas como comunicação ou memória dos processos também podem ser considerados para tratar sistemas distribuídos[10, 11]. Ainda, um par delas podem agir conjuntamente para oferecer uma solução mais completa[6]. Nesse contexto, a pesquisa atual em reescalonadores para *Grids* consiste em oferecer uma interface simples para migração, bem como unir métricas e trabalhar com adaptações eficientes.

Figura 1. Atuação do Potencial de Migração

Os desafios relacionados anteriormente impulsionaram o desenvolvimento do modelo chamado MigBSP. Ele controla o reescalonamento de processos em ambientes dinâmicos e heterogêneos como *Grids* baseados em Cluster-de-Clusters. Ele atua sobre aplicações BSP (*Bulk Synchronous Parallel*) porque esse estilo representa uma organização comum para a escrita de códigos MPI[5, 1]. A facilidade de migração é oferecida em nível de *middleware* sem a intervenção dos desenvolvedores. O escalonador é ligado junto com a biblioteca de programação, não impondo modificações no código fonte da aplicação. A principal contribuição de MigBSP se revela na sua função de decisão denominada Potencial de Migração (PM). PM mede a carga dos processos e é usado para a seleção deles no momento da migração. PM é calculado através da combinação das métricas computação, comunicação e memória. A combinação delas cria uma força resultante que decidirá sobre a movimentação de um processo.

Os vetores de comunicação e computação trabalham a favor da migração, enquanto o de memória participa como uma força de atrito. A Figura 1 ilustra a abordagem de PM,

que representa nossa principal estratégia para lidar com o dinamismo em nível de aplicação e recursos. Além dele, MigBSP também fornece o controle no lançamento do reescalonamento de acordo com o estado do sistema. Isso contribui para a redução de seu próprio impacto na execução da aplicação BSP. Esse artigo descreve MigBSP em detalhes, enfatizando a sua contribuição na combinação de múltiplas métricas. MigBSP foi simulado sobre uma plataforma compartilhada e outra dedicada e guiou o remapeamento de processos de uma aplicação BSP *CPU-bound* que realiza a compressão de imagens[9]. A próxima seção apresenta o referido modelo de reescalonamento e as razões do uso de suas três métricas de atuação.

2 MigBSP: Modelo de Reescalonamento

Esquemas de escalonamento para sistemas paralelos multi-programados podem ser vistos em dois níveis. No primeiro, processadores são alocados para um job. MigBSP é incluído no segundo nível, no qual processos de um job são (re)escalonados usando a lista de processadores. As primeiras ideias de MigBSP podem ser conferidas em [4]. A referida publicação não descreve as razões para a combinação de múltiplas métricas e o desempenho de MigBSP em ambientes compartilhados. MigBSP é executado em uma arquitetura montada com Conjuntos e Gerentes de Conjuntos. Um Conjunto pode representar um *cluster*, uma rede local ou, ainda, um supercomputador.

2.1 Analisando o Uso de Várias Métricas

A principal métrica para o balanceamento de carga em sistemas distribuídos está relacionada a questões de computação. Ela pode ser vista como o tempo gasto em cada processo ou tarefa para executar um conjunto de instruções. Com o objetivo de demonstrar um possível problema nesta abordagem, apresenta-se uma infraestrutura hipotética com dois *clusters* denominados de Cluster1 e Cluster2 e uma situação de migração entre eles. Ambos possuem 10 nós, cada qual com um processador, e uma conexão intra-cluster de 1Gbit/s. A rede entre os dois *clusters* é caracterizada por 10Mbits/s. Cluster1 tem processadores de 500MHz, enquanto Cluster2 tem nós com 1GHz.

Supondo que o mapeamento inicial atribui todos os seis processos para o Cluster1, é possível projetar um cenário onde o processo $p1$ é escolhido para migrar de Cluster1 para Cluster2. A Figura 2 ilustra ambas situações: (a) antes do reescalonamento de $p1$ e; (b) após sua migração. A transferência pode resultar em uma aplicação lenta, uma vez que todas as comunicações de $p1$ devem passar por uma conexão lenta que existe entre os *clusters*. Assim, aplicações *IO-bound* podem sofrer penalidades de desempenho com esta abordagem de escalonamento[11]. Agora, supondo que $p1$ está sendo executado em Cluster2 enquanto os demais permanecem em Cluster1. Além disso, temos uma conexão Gigabit Ethernet entre os *clusters* neste cenário (ver Figura 3(a)). Se a migração de $p1$ de Cluster2 para Cluster1 ocorrer, vamos verificar que o seu tempo para concluir a fase de computação dobrou se comparado com o tempo inicial. Embora as comunicações tornem-se mais rápidas através da aproximação dos processos, o tempo de $p1$ irá comprometer o tempo da aplicação como um todo.

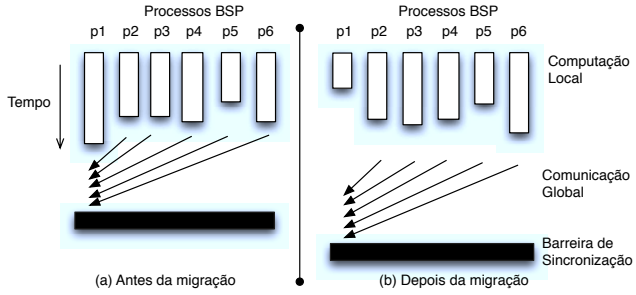

Figura 2. Migração de $p1$ do Cluster1 (lento) para Cluster2 (rápido) usando uma rede lenta

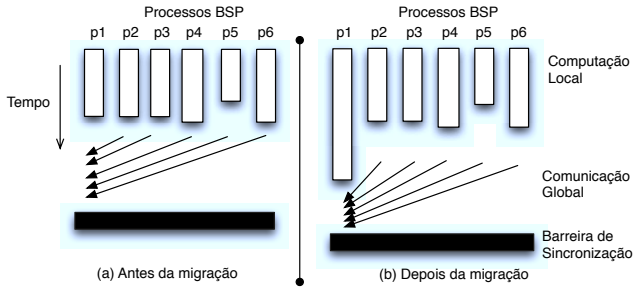

Figura 3. Considerando uma rede rápida, $p1$ é realocado de Cluster2 para Cluster1

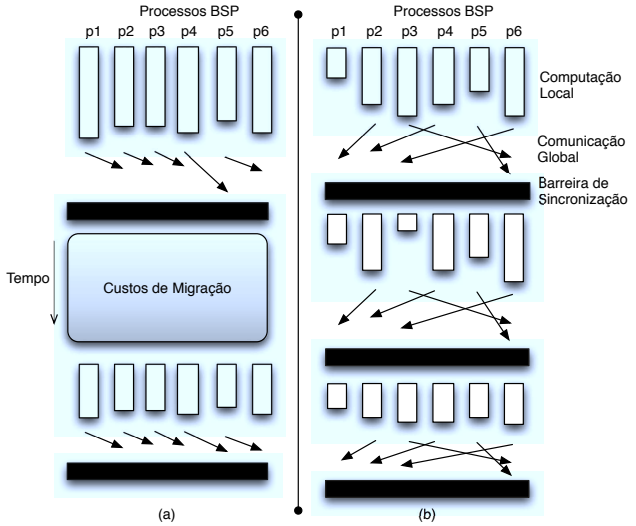

Figura 4. Observação do custo de migração (a) e do comportamento (b) dos processos

Uma estratégia de balanceamento de carga mais refinada pode empregar estratégias de computação e comunicação em sua função de decisão. Considerando isso, aplicações BSP podem ser executadas de forma mais rápida em ambientes heterogêneos uma vez que ambas as partes de uma superetapa são consideradas na redistribuição dos processos. Apesar desta sentença ser verdadeira, a migração de processos pode se tornar facilmente um gargalo (ver Figura 4(a)). Estratégias e ferramentas de migração, memória dos processos e características da rede são as principais causas desta sobrecarga. Neste contexto, um número maior de superetapas pode ser exigido com o objetivo de amortizar os custos de migração quando o reescalonamento traz benefícios e o melhor aproveitamento dos recursos.

A organização em fases permite detectar padrões como apresentado na Figura 4(b). Uma estratégia pode ser empregada como segue: quanto mais estável o comportamento de um processo, maiores são as chances para migrá-lo. Esta abordagem faz uma aposta que um processo específico continuará seu comportamento atual no recurso de destino. $p3$ na Figura 4(b) apresenta uma grande variação em sua fase de cálculo, reduzindo as chances de sua migração.

2.2 Oferecendo Tratamento Dinâmico para o Reescalonamento de Processos

Basicamente, MigBSP oferece estratégias em dois níveis para lidar com a dinâmica do ambiente: (i) na escolha dos candidatos à migração e; (ii) no controle adaptativo do intervalo entre o reescalonamento de processos. A abordagem PM funciona no primeiro nível. Cada processo i computa n funções $PM(i,j)$, onde n é o número de Conjuntos e j significa um Conjunto específico. As razões por trás de PM consistem na realização de apenas um subconjunto de testes de processos para recursos no momento do reescalonamento. PM é expresso pela Equação 1. Suas partes podem ser vistas nas Equações 2, 3 e 4. Quanto maior o $PM(i,j)$, maiores são as chances de migração do processo.

$$PM(i,j) = Comp(i,j) + Comm(i,j) - Mem(i,j) \quad (1)$$

$$Comp(i,j) = P_{comp}(i) \,.\, CTP(i) \,.\, ISet(j) \quad (2)$$

$$Comm(i,j) = P_{comm}(i,j) \,.\, BTP(i,j) \quad (3)$$

$$Mem(i,j) = M(i) \,.\, T(i,j) \;+\; Mig(i,j) \quad (4)$$

$P_{\text{comp}}(i)$ mede a estabilidade do processo i em relação ao número de instruções em cada superetapa. Este valor é próximo de 1 se o processo é regular e próximo de 0 caso contrário. Este parâmetro é dependente somente do comportamento da aplicação. Por exemplo, $p6$ na Figura 4 tem P_{comp} próximo de 1, enquanto $p3$ apresenta seu índice próximo de 0. Outro elemento em $Comp(i,j)$ é a predição do tempo de computação $CTP(i)$ do processo i na superetapa $k+\alpha-1$. k significa a primeira superetapa após a última chamada do reescalonamento e α o intervalo para a próxima chamada. Supondo que $CT_t(i)$ é o tempo de computação do processo i durante a superetapa t, então a predição $CTP(i)$ é calculada como segue.

$$CTP_t(i) = \begin{cases} CT_t(i) & if\ t = k \\ \frac{1}{2}CTP_{t-1}(i) + \frac{1}{2}CT_t(i) & if k < t \leq k+\alpha-1 \end{cases} \quad (5)$$

$CTP_{k+\alpha-1}(i)$ é calculado em unidades de tempo. Seu valor depende da carga da aplicação e dos processadores. Da mesma forma, a métrica de Comunicação também usa predição de tempo e um padrão $P_{comm}(i,j)$ que atua sobre as ações de troca de mensagens do processo i para o Conjunto j. Por fim, $Mem(i,j)$ leva em consideração o espaço na memória do processo, o tempo de transferência entre Conjuntos e os custos de migração. Este último item pode incluir, por exemplo, a reorganização das conexões, a serialização da memória, *checkpoint* e o tempo gasto para criar outro processo no nó de destino. As três métricas de PM tratam da dinamicidade em nível de infraestrutura e aplicação (ver Tabela 1). Uma vez que a aplicação não é alterada, essas métricas representam uma chance para obter melhor desempenho de uma maneira fácil na visão do programador.

Uma vez que o mapeamento dinâmico de processos incorre uma sobrecarga no tempo de execução não negligenciável, outro problema crítico consiste em responder quando migrar. O tratamento da migração não pode ser compensado pela sua sobrecarga. MigBSP avalia migrações entre duas superetapas usando remapeamentos periódicos. O remapeamento a cada k etapas tem sido aplicado a muitas aplicações paralelas. Em particular, esta técnica é usada em aplicações que exibem alteração gradual na carga de trabalho[7]. MigBSP usa uma variável denotada α que é atualizada a cada chamada de reescalonamento, indicando o intervalo da próxima. Primeiramente, as variações no estado do sistema devem ser capturadas. Para tanto, uma variável chamada α' é usada e atualizada a cada superetapa através do incremento ou decremento de uma unidade. α' é passado para α após a ativação da migração.

Com o objetivo de gerar menos intrusividade possível na aplicação, nós aplicamos duas adaptações sobre α ($\alpha \in \mathbb{N}^*$). Basicamente, os objetivos das adaptações são: (i) postergar chamadas ao reescalonamento se os processos estão balanceados e torná-las mais frequentes, caso contrário; (ii) adiar esta chamada se um padrão sem migrações em ω chamadas é observado. Uma variável denotada D é usada para indicar a percentagem de quão longe o processo mais lento e o mais rápido podem estar a partir da média para considerar os processos balanceados. Em suma, quanto maior o valor de α, menor o impacto do modelo sobre a execução da aplicação. Em termos de implementação, os processos salvam o tempo de suas superetapas em um vetor e passam eles para seus Gerentes de Conjunto quando o reescalonamento

Tabela 1. Métricas usadas no cálculo de PM e o impacto de seus parâmetros quanto a dinamicidade

Métricas	Função	Origem do Dinamismo	Explicação
Computação	$CTP(i)$	Aplicação e infraestrutura	Predição do tempo de computação do processo i
Computação	$P_{comp}(i)$	Aplicação	Padrão de Computação do processo i
Computação	$ISet(j)$	Infraestrutura	Índice de desempenho do Conjunto j considerando todos os Conjuntos do ambiente
Comunicação	$BTP(i,j)$	Aplicação e infraestrutura	Predição do tempo de comunicação considerando ações de comunicação entre o processo i e todos os processos do Conjunto j
Comunicação	$P_{comp}(i,j)$	Aplicação	Padrão de Comunicação do processo i e todos os processos do Conjunto j
Memória	$M(i)$	Aplicação e infraestrutura	Espaço em memória ocupado pelo processo i
Memória	$T(i,j)$	Infraestrutura	Tempo para transferir 1 byte do Gerente do processo i para o Gerente do Conjunto j
Memória	$Mig(i,j)$	Infraestrutura	Tempo gasto em operações de migração

é ativado. Depois disso, todos os Gerentes de Conjunto trocam seus vetores. Gerentes de Conjunto tem os tempos de cada processo BSP e calculam a situação do balanceamento. Assim, cada Gerente conhece a variação de α' localmente.

2.3 Escolhendo os Candidatos à Migração

Processos BSP calculam $PM(i,j)$ localmente. A cada chamada de reescalonamento, cada processo passa o seu maior $PM(i,j)$ para o seu Gerente de Conjunto. Esta última entidade troca o PM de seus processos com outros Gerentes. Considerando isso, cada Gerente de Conjunto usa a ideia de lista de escalonamento com o objetivo de selecionar os candidatos à migração criando uma lista ordenada decrescente baseada no maior PM de cada processo BSP. Esta lista é utilizada para aplicar uma das duas possíveis heurísticas. A primeira heurística escolhe processos que possuem PM maior que $Max(PM).x$, onde $Max(PM)$ é o maior PM e x uma percentagem. A segunda heurística pega um processo, o primeiro da lista, que tem o maior PM.

Antes de qualquer migração, a sua viabilidade é verificada considerando os seguintes dados: (i) a carga dos processadores de origem e destino; (ii) os processos BSP que ambos os processadores estão executando; (iii) a simulação de execução do processo no processador destino; (iv) o tempo das ações de comunicação considerando os processadores origem e destino e; (v) custos de migração. Assim, nós calculamos dois tempos: t_1 e t_2. t_1 significa a execução local do processo i, enquanto t_2 abrange a sua execução no outro processador e inclui os custos. Para cada candidato, um novo recurso é escolhido se $t_1 > t_2$.

3 Modelagem da Aplicação BSP

MigBSP será avaliado com uma aplicação baseada em programação dinâmica (DP)[9]. Em particular, observou-se o algoritmo de Smith-Waterman, que é um método conhecido para o alinhamento local de sequências. O algoritmo de Smith-Waterman procede em uma série de frentes de onda em diagonal do outro lado da matriz. A Figura 5(a) ilustra a concepção do algoritmo para uma matriz 5×5 com uma coluna baseada em alocação de processo. Quanto mais intenso o sombreamento, maior a densidade da computação da célula. Cada frente de onda tem a sua própria carga e corresponde a uma superetapa (Figura 5 tem 9 superetapas).

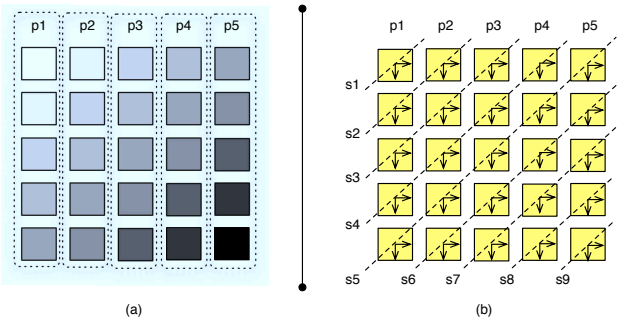

Figura 5. (a) Escalonamento em colunas; (b) Mapeamento de processos nas superetapas

A organização de mapeamento baseada na superetapa diagonal e o mapeamento dos processos em colunas trazem as seguintes conclusões: (i) $2n-1$ superetapas são cruzadas para calcular uma matriz quadrada de ordem n e; (ii) cada processo estará envolvido em n superetapas. A Figura 5(b) mostra as ações de comunicação entre os processos. Considerando que a célula x, y (x e y significam a linha e a coluna da matriz, respectivamente) necessita de dados de $x, y-1$ e $x-1, y$, teremos uma interação do processo py para o processo $py+1$. Não há a comunicação dentro da mesma coluna, uma vez que corresponde ao mesmo processo.

4 Metodologia de Avaliação

O objetivo da avaliação é observar o comportamento de MigBSP em resposta a dinamicidade da aplicação e dos recursos. Para tal, aplicou-se simulação em três cenários: (i) uso da aplicação simplesmente; (ii) aplicação com MigBSP sem aplicar migrações e; (iii) aplicação com MigBSP com migrações. Tanto a aplicação quanto MigBSP foram desenvolvidos usando o SimGrid[3]. SimGrid é determinístico, onde uma entrada específica resulta sempre na mesma saída.

Com o objetivo de testar os cenários, nós montamos uma infraestrutura com cinco Conjuntos conforme ilustrado na Figura 6. Um Conjunto representa um *cluster* onde cada nó possui um único processador. Os testes foram executados com dois valores iniciais para α, 4 e 16, e D igual

a 0.5. Tais valores foram escolhidos empiricamente. Nós observamos o comportamento de 10, 25, 50, 100 e 200 processos. Nós optamos em utilizar uma abordagem contínua onde um *cluster* é totalmente preenchido antes de passar para o próximo[12]. A Figura 7 mostra o mapeamento de 200 processos. Da mesma forma, apenas os *clusters* A, B, C e D serão usados na execução de 50 processos. O número de processos é igual à dimensão da matriz.

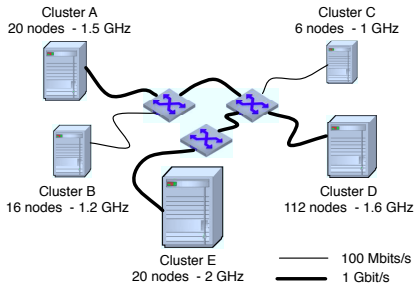

Figura 6. Ambiente de execução. Tamanho de cada *cluster* depende de seu *clock*

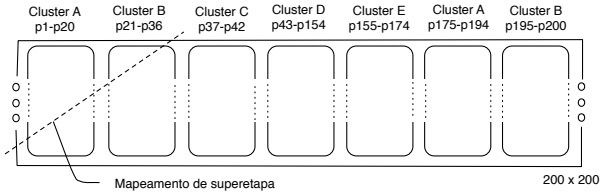

Figura 7. Mapeamento de 200 processos

Foi empregado o valor de 10^6 como o número de instruções para a primeira superetapa e 10^9 para a última. Trabalhou-se com matrizes quadradas de ordem 10, 25, 50, 100 e 200. Cada célula de uma matriz 10×10 necessita de 500 Kbytes para se comunicar e cada processo consome 1.2 Mbytes de memória (700 Kbytes compreendem outros dados da aplicação). A célula da matriz 25×25 comunica 200 Kbytes e cada processo ocupa 900 Kbytes em memória e assim por diante. Por fim, os custos de migração são baseados em execuções com AMPI [8] em nossos *clusters*.

5 Avaliação de MigBSP

O dinamismo da aplicação ocorre na variação da carga ao longo das superetapas. Já o dinamismo de recurso foi modelado com a redução do poder dos *clusters* D e E pela metade após x segundos de execução da aplicação. Essa estratégia visa criar uma situação de compartilhamento de recursos análoga a que acontece em *Grids*. A estratégia tende a mapear processos para os *clusters* mais rápidos. No entanto, eles se tornarão os mais escolhidos depois de x segundos. Neste ponto, MigBSP deve lidar com a perda de desempenho sugerindo futuras migrações viáveis e eficientes. A Figura 8 ilustra a estratégia de dinamismo na infraestrutura. As próximas seções apresentam as duas situações de dinamismo. A Subseção 5.1 lida apenas com o dinamismo de aplicação, enquanto a Subseção 5.2 mostra os resultados quando permitindo dinamismo de aplicação e infraestrutura. Ambos utilizam a notação p_n para denotar o n^{th} processo no escalonamento inicial da aplicação.

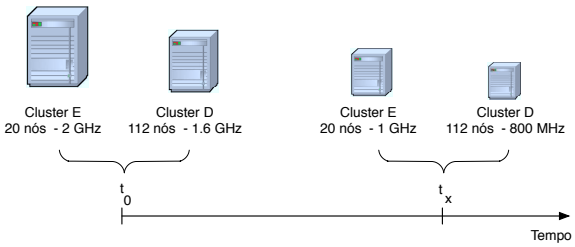

Figura 8. Aplicação possui 50% do poder dos *clusters* D e E após x segundos

5.1 Dinamismo em Nível de Aplicação

A Figura 9 apresenta o desempenho de MigBSP quando habilitado migrações. Um ganho médio de 9,58% foi alcançado com o valor inicial de α igual a 4 e 16. Por exemplo, ganhos de 12,2% e 5,6% foram observados durante a execução de 50 e 200 processos, respectivamente. O valor supracitado de 9,58% é explicado através do comportamento da aplicação, no qual a carga aumenta ao longo das superetapas. Especialmente, processos dos *clusters* A e B são migrados para D e E no início da aplicação. Os *clusters* D e E são os mais rápidos e estão inativos na primeira superetapa. No entanto, ambos recebem processos que vão iniciar mais tarde, simultaneamente com os migrados anteriormente. A Figura 10 retrata a sobrecarga de MigBSP comparando os cenários i e ii. Quanto maior o tamanho da matriz, menor é a sobrecarga durante a execução da aplicação.

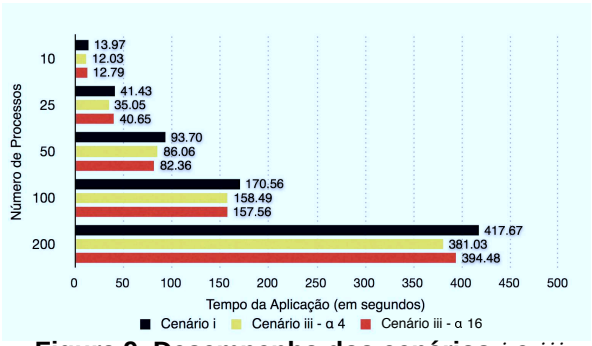

Figura 9. Desempenho dos cenários i e iii

A Figura 11 explica a execução de 399 superetapas com o cenário iii e 200 processos. O reescalonamento sobre as primeiras 6 chamadas não foi viável devido a baixa carga de computação envolvida nesta superetapa. Especialmente, as primeiras quatro tentativas resultaram em valores negativos de PM devido ao menor peso em ações de computação e comunicação se comparado aos custos de migração. O

primeiro rearranjo ocorre na superetapa 44, onde 6 processos do *cluster* mais lento C são passados para executar mais rápido em E. Neste momento, este último *cluster* é inativo e seus 6 processadores inicias são ocupados. A superetapa 76 é marcada por 14 migrações: p_{23} a p_{36} são passados para o *cluster* E. Este movimento preenche o número de um processo trabalhando por nó no *cluster* mais rápido. p_{21} e p_{22} permanecem no *cluster* B pois possuem menos carga para computar em relação a coleção migrada na superetapa 76.

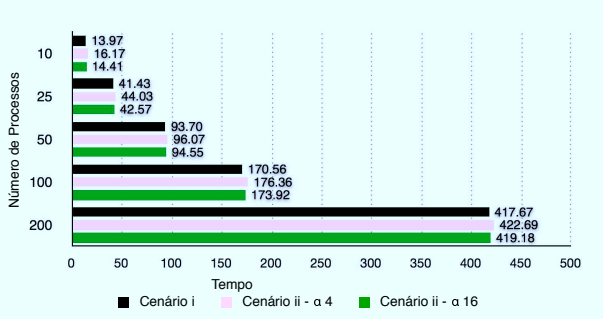

Figura 10. Sobrecarga imposta por MigBSP

Seguindo a análise da Figura 11, 14 processos são migrados na superetapa 140 para o *cluster* D. Esta ação desencadeia a migração de p_{21} e p_{22}, que executam as superetapas mais pesadas até chegar na superetapa 140. Além disso, o *cluster* D recebe os primeiros 12 processos da aplicação. Outros processos no *cluster* A também foram testados para migração. p_{13} até p_{20} foram analisados mas não migraram uma vez que os *clusters* D e E estavam com a carga de trabalho lotada. p_{195} até p_{200} (nós restantes no *cluster* B), assim como p_{175} até p_{194} (processos mapeados para o *cluster* A) foram movidos para o *cluster* D na superetapa 268. Este *cluster* foi escolhido porque tinha recursos ociosos no momento. Por fim, é importante observar que os *clusters* D e E receberam praticamente todos os processos. No entanto, eles têm no máximo um processo por nó na parte final da aplicação (após a superetapa 268). Inicialmente, o primeiro processo mapeado para o *cluster* D começa o seu trabalho na superetapa 43. Assim, ele tem nós livres na superetapa 268, pois a diferença de 268 e 200 (número total de superetapas por processo) é maior que 43. Especialmente, a subtração de 68 por 43 significa o número de processos que foram migrados para o *cluster* D na superetapa 268.

5.2 Dinamicidade em Nível de Aplicação e Recursos

Considerando que a aplicação é *CPU-bound*, seu início é marcado por uma fase de migração de processos mais lentos para os *clusters* mais rápidos. Depois de reduzir o poder de processamento dos *clusters* D e E, uma nova rodada de migrações ocorre na ordem inversa. A Tabela 2 apresenta os resultados quando a aplicação sente a variação de desempenho depois de passados 10 segundos. Processos p_1, p_2 e p_3 são passados para o *cluster* E após a superetapa 4 em testes com 25 processos. Este *cluster* recebe p_4 até p_{10} na superetapa 12. Esta superetapa ocorre após 3,18 segundos de execução. A próxima chamada de migração ocorre na superetapa 28 onde a aplicação dura 13,61 segundos. Ela é marcada pelo retorno dos processos p_3 até p_{10} para o *cluster* A. Em adição, os primeiros 3 processos do *cluster* B são transferidos para A durante este ponto do reescalonamento. Seguindo as decisões de PM, p_1, p_2 e p_3 permanecem em E, uma vez que foram responsáveis por superetapas leves.

Tabela 2. Resultados após a variação de desempenho depois de passados 10 segundos

Cenário	Processos				
	10	25	50	100	200
iii com α 4	14.04	41.34	102.50	257.58	579.34
iii com α 16	14.43	42.98	102.95	263.46	572.82
i	15.17	45.32	124.56	269.29	564.79

A Figura 12(a) apresenta a execução de 50 processos que variam o tempo t_x onde os *clusters* D e E mudam suas capacidades de processamento. Usando t igual a 25 e α igual a 4, p_1 e p_2 migram do *cluster* A para E na superetapa 4. Além disso, a superetapa 12 é marcada pela migração de p_3 até p_9 para este último *cluster*. O próximo conjunto de migrações ocorre na superetapa 28, onde os primeiros 4 processos do *cluster* B são passados para E. Enquanto a chamada na superetapa 28 acontece com 8,68 segundos, a próxima é feita ao passar 40,99 segundos na superetapa 60. Esta última chamada apresenta 8 migrações: p_{43} até p_{50} para o *cluster* A. O *cluster* A representa um alvo adequado para migração quando D e E estão sobrecarregados. O resultado satisfatório com este valor de t é explicado porque o primeiro escalonamento não mapeia os processos para o *cluster* E. Assim, os processos que ele recebe finalizam as suas computações mais rápido (antes de 25 segundos) desde a primeira superetapa com uma parte levemente carregada da matriz. Ao mudar t para 75 e α para 16, 13,33% de ganho são observados com MigBSP. Este contexto permite migrações nas superetapas 16 e 48. No primeiro caso, p_1 até p_{15} foram transferidos para o *cluster* E. A superetapa 16 foi tomada com 2,2 segundos do tempo de execução da aplicação. A segunda chamada é responsável pela migração de processos do *cluster* C para A. MigBSP escolheu o *cluster* A ao invés de E porque o *cluster* E tem somente 5 nós livres. Assim, o *cluster* A recebe os 6 processos que podem rodar juntos em um único *cluster* e mais rápido.

A Figura 12(b) ilustra a atividade de 100 processos. Especialmente, quando t_x é igual a 100 e α igual a 4, a superetapa 10 migra 7 processos do *cluster* A para E. A superetapa 10 é responsável pela migração dos próximos 11 processos do *cluster* A e o primeiro processo de B para E. Todos os processos do *cluster* C foram passados para A após a execução da superetapa 46. Na superetapa 94, os 14 primeiros processos do *cluster* D foram passados para o A

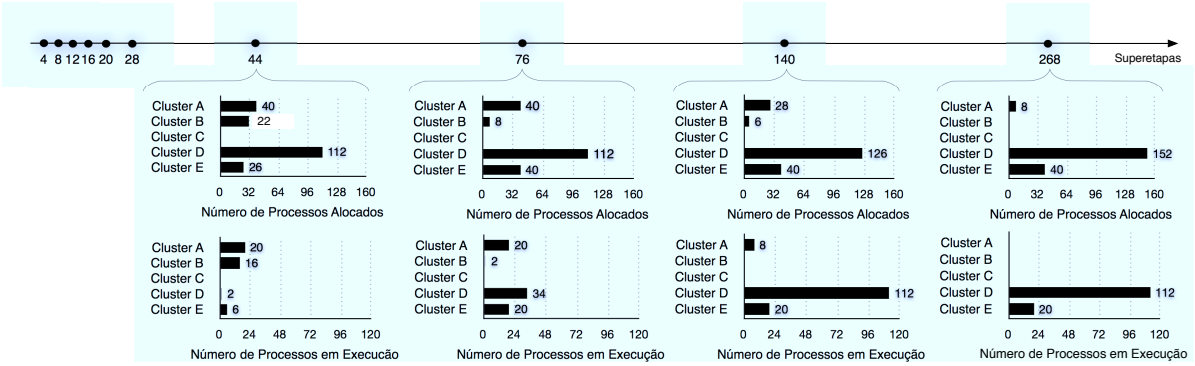

Figura 11. Execução de 200 processos com 10 chamadas de migração, sendo que 4 resultaram em transferências. Os gráficos mostram a alocação e o uso de fato de cada *cluster*

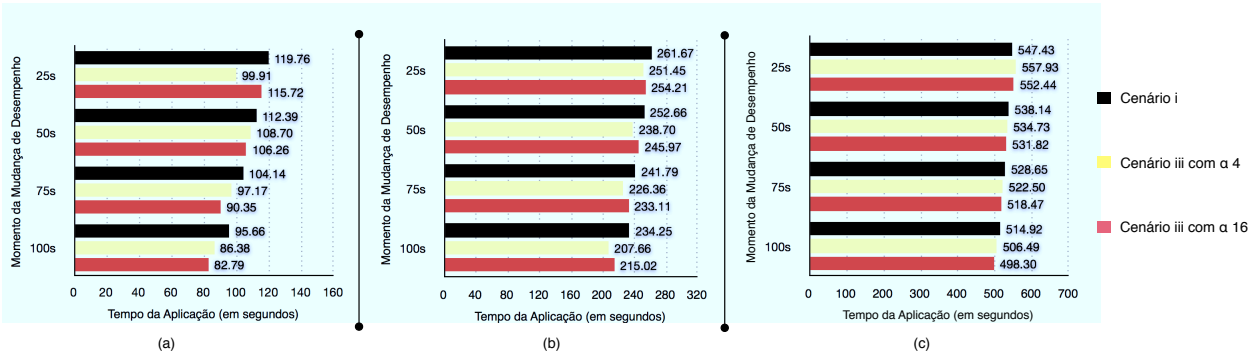

Figura 12. Testes de 50 (a), 100 (b) e 200 (c) processos com dinamicidade da aplicação e de recursos

também. Em adição, a superetapa 190 escolhe 10 processos (p_{91} até p_{100}) do *cluster* D para a execução mais rápida em A. A dinamicidade da aplicação explica o fato do *cluster* A receber mais processos. No final da aplicação, os processos previamente migrados para A não estão em execução e os recursos estão livres, permitindo as migrações.

Outra situação pertinente é a execução de 100 processos com t igual a 25 segundos e α igual a 16. A superetapa 16 ocorre com 1,95 segundos e representa o ponto onde 14 processos do *cluster* A são transferidos para E. A próxima chamada ocorre na superetapa 46, onde todos os processos do *cluster* C são passados para A. Neste ponto, a aplicação executou por 12,87 segundos. A superetapa 68 denota o ponto onde 25 segundos foram alcançados. Depois, uma alternativa para obter desempenho consiste em chamadas mais frequentes para migração. No entanto, o intervalo para a próxima chamada de migração foi atualizado na superetapa 46 e vai ocorrer na superetapa 106. Este último ponto mostra a migração dos 9 primeiros processos do *cluster* D para A. Apesar de adiar a chamada de migração, MigBSP obtém um ganho de 4% ao comparar os cenários i e iii.

A Figura 12(c) apresenta o conjunto de testes considerando recursos não dedicados e 200 processos. Com t_x igual a 25 e α igual a 4, as primeiras 4 superetapas não apresentam migrações. As chamadas acontecem nas superetapas 4, 8, 12 e 16. A situação de desequilíbrio é expressa no intervalo entre as chamadas, que permanece constante no início da aplicação. A superetapa 20 é caracterizada pela migração de 18 processos do *cluster* A para E. A superetapa 24 move 2 processos de B para E. Após 6 chamadas de escalonamento, os processos tornam-se balanceados e α começa a crescer indicando a próxima chamada para a superetapa 30. Esta superetapa é responsável por mover 6 processos do *cluster* B para A. Em adição, ela move 2 processos do *cluster* E (aqueles previamente alocados para B) para A. Este movimento é dirigido pela força de comunicação. As superetapas 42 e 66 migram todos os processos do *cluster* C para A. A primeira move 4 enquanto as outras movem 2. Este ponto marca a execução de 25 segundos de aplicação. A superetapa 114 transfere 3 processos do *cluster* E para B. O desempenho é limitado pelo fato de que a próxima chamada de migração é estabelecida antes da variação da capacidade de processamento dos *clusters* D e E.

6 Trabalhos Relacionados

O *framework* ReSHAPE emprega tentativa-e-erro para o reescalonamento de processos. Embora possibilite gerenciar a infraestrutura em tempo real, este método é demorado e pode envolver o reescalonamento para configurações ineficientes de processadores[14]. Vadhiyar e Dongarra

apresentaram um *framework* de migração e auto-adaptação no sistema GrADS[15]. O ganho com reescalonamento é baseado na predição do tempo de execução restante sobre um novo recurso. Este *framework* deve trabalhar com aplicações com partes e durações conhecidas com antecedência. O modelo proposto por Boukerche considera as métricas de computação e comunicação em uma fórmula que é usada para determinar os desequilíbrios de carga baseado em um algoritmo de *simulated annealing*[5].

Considerando o contexto de migração, as bibliotecas PUBWCL [2] e PUB [1] permitem esta facilidade em aplicações BSP. Os algoritmos propostos usam dados sobre os tempos de computação de cada processo, bem como dados em relação à carga dos nós. Em particular, a PUB apresenta estratégias centralizadas e distribuídas para o balanceamento de carga. Na abordagem distribuída, cada nó escolhe c outros nós de forma aleatória e os questiona sobre a sua carga. Um processo é migrado se a carga mínima entre os c nós analisados é menor que a carga do nó que faz o teste. Ambas as estratégias não consideram a comunicação entre os processos, nem os custos de migração.

7 Conclusão

Esse artigo apresentou MigBSP, destacando o comportamento de suas métricas sobre uma aplicação em *Grid*. Sua contribuição aparece em sua abordagem para lidar com aplicações e recursos dinâmicos. A função PM indica o ajuste de carga através da migração dos processos que possuem longos tempos de computação, realizam ações de comunicação com outros processos que pertencem a um mesmo local (por exemplo, um *cluster*) e apresentam custos de migração baixos. Os resultados mostram ganhos de migração de até 16,9%. Em adição, MigBSP revelou uma sobrecarga média de 5,4% quando migrações não ocorrem. PM contribui para isso, pois considera os processos e Conjuntos ao invés de processos e processadores.

Os resultados foram satisfatórios uma vez que modificações não são necessárias na aplicação, tampouco execuções extras para a captura de dados de balanceamento de carga são requeridos. Por fim, trabalhos futuros incluem o uso de MigBSP no contexto da computação em nuvem. Ao invés de alocar novos recursos, ele pode ser usado para realocar os processos para quando o acordo de execução de um determinado serviço for desfeito.

Agradecimentos

Este trabalho foi parcialmente financiado pelos órgãos: CAPES, CNPq e RNP CTIC.

Referências

[1] O. Bonorden. Load balancing in the bulk-synchronous-parallel setting using process migrations. In *21th International Parallel and Distributed Processing Symposium (IPDPS 2007)*, pages 1–9. IEEE, 2007.

[2] O. Bonorden, J. Gehweiler, and F. M. auf der Heide. Load balancing strategies in a web computing environment. In *Int. Conf. on Parallel Processing and Applied Mathematics (PPAM)*, pages 839–846, Poland, 2005.

[3] H. Casanova, A. Legrand, and M. Quinson. Simgrid: A generic framework for large-scale distributed experiments. In *Tenth International Conference on Computer Modeling and Simulation (uksim)*, pages 126–131, Los Alamitos, CA, USA, 2008. IEEE Computer Society.

[4] R. da Rosa Righi, L. L. Pilla, A. Carissimi, P. Navaux, and H.-U. Heiss. Migbsp: A novel migration model for bulk-synchronous parallel processes rescheduling. *High Performance Computing and Communications, 10th IEEE International Conference on*, 0:585–590, 2009.

[5] R. E. De Grande and A. Boukerche. Dynamic balancing of communication and computation load for hla-based simulations on large-scale distributed systems. *J. Parallel Distrib. Comput.*, 71:40–52, January 2011.

[6] C. Du, X.-H. Sun, and M. Wu. Dynamic scheduling with process migration. In *CCGRID '07: Proceedings of the Seventh IEEE International Symposium on Cluster Computing and the Grid*, pages 92–99, 2007. IEEE.

[7] N.-T. Fong, C.-Z. Xu, and L. Y. Wang. Optimal periodic remapping of dynamic bulk synchronous computations. *J. Parallel Distrib. Comput.*, 63:1036–1049, November 2003.

[8] C. Huang, G. Zheng, L. Kale, and S. Kumar. Performance evaluation of adaptive mpi. In *PPoPP '06: Proc. of the eleventh ACM SIGPLAN symposium on Principles and practice of parallel programming*, pages 12–21, 2006. ACM Press.

[9] M. Y.-H. Low, W. Liu, and B. Schmidt. A parallel bsp algorithm for irregular dynamic programming. In *Advanced Parallel Processing Technologies, 7th International Symposium*, volume 4847 of *Lecture Notes in Computer Science*, pages 151–160. Springer, 2007.

[10] D. S. Nikolopoulos and C. D. Polychronopoulos. Adaptive scheduling under memory constraints on non-dedicated computational farms. *Future Gener. Comput. Syst.*, 19:505–519, May 2003.

[11] J. M. Orduna, V. Arnau, A. Ruiz, R. Valero, and J. Duato. On the design of communication-aware task scheduling strategies for heterogeneous systems. In *ICPP '00: Int. Conf. on Parallel Processing*, page 391, 2000. IEEE.

[12] J. A. Pascual, J. Navaridas, and J. Miguel-Alonso. Job scheduling strategies for parallel processing. chapter Effects of Topology-Aware Allocation Policies on Scheduling Performance, pages 138–156. Springer-Verlag, Berlin, 2009.

[13] H. Sanjay and S. Vadhiyar. Strategies for rescheduling tightly-coupled parallel applications in multi-cluster grids. *Journal of Grid Computing*, pages 1–25, 2010. 10.1007/s10723-010-9170-z.

[14] R. Sudarsan and C. Ribbens. Reshape: A framework for dynamic resizing and scheduling of homogeneous applications in a parallel environment. In *Parallel Processing, 2007. ICPP 2007. Int. Conference on*, page 44, sept. 2007.

[15] S. S. Vadhiyar and J. J. Dongarra. Self adaptivity in grid computing: Research articles. *Concurr. Comput. : Pract. Exper.*, 17(2-4):235–257, 2005.