Avaliação do Re-escalonamento Adaptativo de Processos BSP*

Laércio Lima Pilla[1], Rodrigo Righi[1], Alexandre Carissimi[1], Philippe Navaux[1], Hans-Ulrich Heiss[2]

[1]Instituto de Informática – Universidade Federal do Rio Grande do Sul (UFRGS)
Caixa Postal 15.064 – 91.501-970 – Porto Alegre – RS – Brazil

[2]Kommunikations- und Betriebssysteme - Technische Universität Berlin (TU-Berlin)
Sekretariat EN 6, Einsteinufer 17 – D-10587 - Berlin – Germany

{llpilla, rrrighi, asc, navaux}@inf.ufrgs.br, heiss@cs.tu-berlin.de

Abstract. *Processes migration is presented as a mechanism for runtime load balancing, mainly over heterogeneous and dynamic environments. In this context, we developed a model called MigBSP that controls the processes rescheduling in BSP (Bulk Synchronous Parallel) applications. Such applications are divided in supersteps, each one containing both computation and communication phases, followed by a barrier synchronization. Since the barrier waits for the slowest process, this model focuses in the adjustment of the processes location in order to reduce the supersteps' times. Considering this scope, the main contributions of MigBSP are: (I) combination of multiple metrics to measure the potencial of migration of each process; (II) observation of patterns to analyze the processes' regularity; and (III) adaptivity on rescheduling calls. This paper presents the MigBSP's experimental results over a regular and an irregular application.*

Resumo. *A migração de processos apresenta-se como um mecanismo para o balanceamento de carga durante a execução de aplicações, principalmente em ambientes heterogêneos e dinâmicos. Neste contexto, foi desenvolvido um modelo chamado MigBSP que controla o re-escalonamento de processos em aplicações BSP (Bulk Synchronous Parallel). Tais aplicações são divididas em superetapas, cada qual contendo fases de computação e comunicação, seguidas de uma barreira de sincronização. Tendo em vista que a barreira espera pelo processo mais lento, o modelo objetiva o ajuste da alocação dos processos para reduzir o tempo das superetapas. Considerando esse escopo, as principais contribuições do modelo são: (I) combinação de múltiplas métricas para medir o potencial de migração de cada processo; (II) consideração de padrões para analisar a regularidade dos processos; e (III) adaptatividade no re-escalonamento de acordo com o estado do sistema. Este artigo apresenta resultados experimentais do modelo sobre uma aplicação BSP regular e outra irregular.*

1. Introdução

A tecnologia de *grid* vem ganhando destaque no cenário atual de computação paralela e distribuída. A interligação entre diferentes *clusters* e computadores com alta capacidade de processamento gera ambientes heterogêneos com grande potencial, podendo

*Pesquisa parcialmente apoiada pela Agência CNPq e pela empresa Microsoft

serem usados tanto para o processamento científico quanto para aplicações comerciais. Os constituintes de tais ambientes podem estar localizados tanto dentro de uma mesma organização quanto espalhados globalmente. Com tais características, o escalonamento de processos num ambiente de *grid* torna-se uma tarefa complexa, normalmente demandando esforço do programador ou usuário. Assim, muitas vezes esse último acaba tendo que conhecer detalhes da aplicação e da arquitetura das máquinas envolvidas. Além disso, cada nova aplicação ou recurso disponibilizado requer novas análises.

Tendo em vista que o desempenho da aplicação está sujeito ao escalonamento dos processos, o uso de balanceamento de carga em nível de *middleware*, ligado à aplicação através de bibliotecas, busca reduzir os problemas do usuário. Por exemplo, um esquema de alocação onde os processos mais lentos são mapeados para os recursos mais rápidos pode ser utilizado. Porém, essa abordagem não é a melhor para aplicações irregulares (com mudanças nos padrões de comunicação e quantidade de computação) e ambientes dinâmicos (com flutuações na carga e disponibilidade dos processadores e rede). Em tais situações, um bom mapeamento inicial pode não se manter eficiente com o tempo [Aggarwal et al. 2003, Huang et al. 2006, Low et al. 2007]. Assim, uma alternativa possível é o re-escalonamento dinâmico de processos durante a execução da aplicação.

Nesse contexto, foi desenvolvido o modelo **MigBSP** que **controla o re-escalonamento de processos** em ambientes heterogêneos e dinâmicos. MigBSP trabalha com aplicações do tipo BSP (*Bulk Synchronous Parallel*) [García and Morales-Luna 2008, Valiant 1990] e utiliza o conceito de hierarquia para reduzir sua complexidade. Para isso, ele trabalha com uma organização em dois níveis baseada em Conjuntos (diferentes redes de computadores) e Gerentes de Conjuntos. O principal objetivo do MigBSP é reduzir o tempo das fases de computação e comunicação dos processos BSP, utilizando informações coletadas durante a execução da aplicação. Tal redução acontece através da migração dos processos mais lentos que apresentam também um baixo custo de migração. Em adição, outra característica de MigBSP é a adaptatividade. Diferentemente de outras abordagens que tratam de aplicações BSP [Bonorden et al. 2005], MigBSP apresenta o re-escalonamento conforme o estado do sistema, de forma a reduzir o impacto do próprio modelo sobre o tempo de execução da aplicação.

Este artigo apresenta o modelo MigBSP e suas idéias para o tratamento de processos BSP. O foco está na medição do impacto do modelo sobre duas aplicações: uma regular e outra irregular. A regularidade está relacionada ao comportamento dos processos a cada superetapa (*superstep*). Para tal avaliação, simula-se uma estrutura *multicluster* com variação no número de processos. Obtiveram-se ganhos de até 15% com o uso do modelo sobre alguns cenários. Além disso, sobrecustos menores que 7% foram observados quando nenhuma migração foi feita. Os resultados mostram a viabilidade do modelo MigBSP, que pode ser utilizado em diversas situações onde o balanceamento de carga se mostra presente, como em servidores Web e computações síncronas em geral [Chen and Tong 2004, Kovacs and Kacsuk 2004].

O artigo é organizado da seguinte forma. A Seção 2 descreve o modelo MigBSP e suas principais características. A metodologia de avaliação é introduzida na Seção 3. As Seções 4 e 5 apresentam os resultados obtidos na simulação das aplicações, o impacto do

modelo e os ganhos de desempenho com seu uso. Trabalhos relacionados são descritos na Sessão 6. Por fim, as conclusões obtidas com esse trabalho, com foco nas contribuições e trabalhos futuros, são apresentadas na Seção 7.

2. MigBSP: Modelo de Re-escalonamento de Processos BSP

MigBSP trata do balanceamento de carga (processos) em aplicações BSP. A realocação dos processos é feita baseada em informações capturadas durante a execução da aplicação. A Figura 1 apresenta uma aplicação BSP irregular em sua superetapa k, onde a carga de processos não está balanceada. Com o re-escalonamento de processos, espera-se que as superetapas seguintes ($> k$) apresentem menores tempos de execução. Entre as formas de se minimizar o tempo total de execução de uma aplicação BSP, o modelo aborda a redução dos tempos de computação e comunicação [Skillicorn et al. 1997]. MigBSP controla a migração de processos BSP e trabalha de acordo com o estabilidade do sistema. Além da computação e comunicação, o modelo considera em seu algoritmo questões como a memória e custos de migração para verificar a viabilidade das migrações.

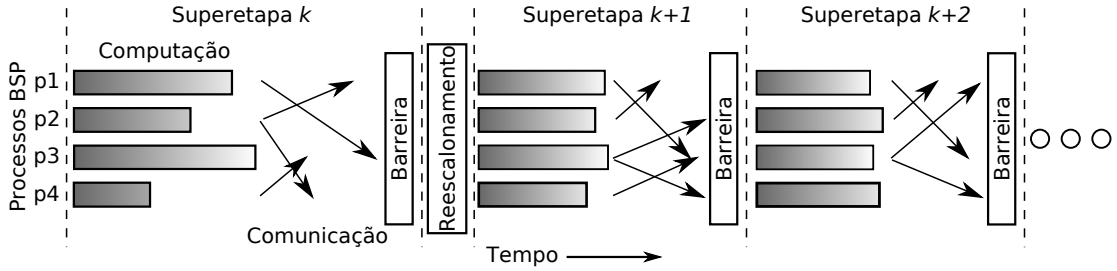

Figura 1. Superetapas em diferentes situações.

O modelo MigBSP prove um formalismo que responde as seguintes questões gerais sobre escalonamento: (I) "Quando" tratar da migração de processos; (II) "Quais" processos são candidatos à migração; e (III) para "Onde" migrar os processos selecionados. MigBSP não trata os mecanismos sobre "Como" efetuar a migração. Em [Righi et al. 2008] foram descritas respostas preliminares para as três questões tratadas pelo modelo. Porém, este trabalho não tratava questões como a escalabilidade do modelo tanto sobre o número de processos quanto de recursos, simulações com aplicações irregulares e tão pouco os conceitos de adaptação no re-escalonamento segundo o estado do sistema.

Segundo a taxonomia de sistemas de escalonamento distribuídos de Casavant e Kuhl [Casavant and Kuhl 1988], MigBSP pode ser incluído nos itens dinâmico e global. A primeira característica se refere à coleta de informações durante a execução. Já a segunda trata sobre tarefas de escalonamento distribuídas. Em adição, o escalonamento final obtido é subótimo e emprega heurísticas.

2.1. Modelo de Ambiente

O modelo MigBSP trabalha sobre um ambiente distribuído, heterogêneo e não dedicado. A heterogeneidade trata de processadores com diferentes velocidades e redes com diferentes larguras de banda - como Fast e Gigabit Ethernet - e níveis - como ambientes *multi-cluster*. O comportamento dinâmico do ambiente está ligado a este ser não dedicado, onde os processos BSP são executados concorrentemente com outros processos, apresentando assim flutuações na carga dos processadores e congestionamentos na rede.

Além disso, o dinamismo pode se apresentar nos próprios processos BSP, os quais podem necessitar de maior poder computacional durante a execução ou alterar suas interações com outros processos. O modelo de máquina inclui multiprocessadores, redes locais, assim como *clusters*. O modelo requer que todos processos possam se comunicar entre si de forma assíncrona. Ainda, o modelo não trata questões de tolerância a falhas.

Os processos BSP são mapeados para processadores. Para tornar o mapeamento mais veloz e flexível, MigBSP utiliza-se de escalonamento hierárquico com o intuito de otimizar a passagem de informações. Essa noção é apresentada no escalonador InteGrade [Goldchleger et al. 2004]. Os nós são reunidos de forma a criar uma abstração de Conjunto. Um Conjunto pode ser uma rede local ou um *cluster*. Cada Conjunto possui um Gerente de Conjunto, o qual captura informações sobre seu Conjunto e as trocas com os outros Gerentes. Mecanismos para a coleta de tais informações são incluídos em todos processos - com código adicional na barreira de sincronização - e nos próprios Gerentes.

2.2. Ativação do Re-escalonamento

O re-escalonamento de processos entra em ação após o final de uma superetapa. Esse ponto ("Quando") foi escolhido porque nele é possível analisar informações sobre a fases de comunicação e computação de todos processos BSP. Isto inclui informações sobre os processos mais lento e mais rápido, assim como o esquema de comunicação entre os processos. Além disso, tal ponto favorece a implementação da migração, uma vez que o sistema se encontra em um estado consistente. De forma a reduzir o impacto do modelo, foram aplicadas duas adaptações. Elas provêm um intervalo dinâmico de superetapas entre duas chamadas de re-escalonamento. Ambas adaptações são apresentadas abaixo.

2.2.1. Primeira Adaptação

O índice α ($\alpha \in N^*$) é utilizado para informar o intervalo de superetapas entre chamadas de re-escalonamento. Esse índice aumenta se o sistema se encontra estável no tempo de conclusão das superetapas e diminui caso contrário. O segundo caso aumenta a frequência de chamadas para tornar o sistema estável mais rapidamente. Para verificar o estado do sistema, capturam-se os tempos de conclusão dos processos a cada superetapa.

$$tempo\ do\ processo\ mais\ lento < tempo\ médio\ .\ (1+D) \quad (1)$$

$$tempo\ do\ processo\ mais\ rápido > tempo\ médio\ .\ (1-D) \quad (2)$$

O sistema é considerado estável se ambas Inequações 1 e 2 forem verdadeiras. Em ambas inequações, D - passado como parâmetro ao modelo - informa em porcentagem a diferença que os processos extremos podem ter da média. O Algoritmo 1 apresenta como o valor de α é calculado durante a execução da aplicação. Uma variável α' é utilizada para guardar o valor temporário de α e pode sofrer variação de uma unidade a cada superetapa.

No Algoritmo 1, t ($k \leq t \leq k+\alpha-1$) é o índice da superetapa e k representa a superetapa que vem após a última chamada de re-escalonamento (ou 1, no início da aplicação). k, α e t possuem o mesmo significado em todos algoritmos. α' possui como limite inferior o valor inicial de α. O modelo busca minimizar sua intrusão na execução da aplicação enquanto o sistema se encontra estável. Vendo pelo lado da implementação do modelo, os processos guardam seus tempos em um vetor e o passam para seus Gerentes quando o re-escalonamento é ativado. Em seguida, os Gerentes de Conjunto trocam informações. Com isso, eles possuem os tempos de todos os processos e podem calcular ambas Inequações 1 e 2. Assim, cada Gerente sabe a variação de α localmente.

Algoritmo 1 Cálculo de α

1: **para** t da superetapa k até a superetapa $k+\alpha-1$ **faça**
2: **se** Inequações 2 e 1 são verdadeiras **então**
3: Aumenta α' de 1
4: **senão se** $\alpha' > \alpha$ inicial **então**
5: Diminui α' de 1
6: **fim**
7: **fim**
8: Chamada ao re-escalonamento de processos BSP
9: $\alpha = \alpha'$

2.2.2. Segunda Adaptação

Outra adaptação trata do gerenciamento de D (ver Inequações 1 e 2) baseado na frequência das migrações. A idéia aqui é aumentar o valor de D se ω chamadas consecutivas de re-escalonamento acontecem mas nenhuma migração é feita. O aumento de D torna maior o intervalo de aceitação do sistema como balanceado, causando consequentemente o aumento de α'. Entretanto, D pode diminuir até seu valor inicial se cada chamada gera a migração de pelo menos um processo.

Algoritmo 2 Cálculo de D

1: $\gamma \leftarrow$ Número de chamadas consecutivas de re-escalonamento sem migrações
2: **se** $\gamma \geq \omega$ **então**
3: D $\leftarrow$ D + $\frac{D}{2}$
4: **senão se** D $>$ D inicial e $\gamma = 0$ **então**
5: D $\leftarrow$ D - $\frac{D}{2}$
6: **fim**

O Algoritmo 2 apresenta como D é gerenciado. Tal computação é feita pelos Gerentes de Conjunto. Essa adaptação é importante principalmente em ambientes onde os custos de migração são altos. Nesse contexto, apesar de processos serem selecionados, eles acabam não sendo migrados e o sistema continua desbalanceado. Assim, o propósito do Algoritmo 2 é minimizar o impacto do modelo em situações como a citada.

2.3. Escolha dos Candidatos à Migração

A função de decisão chamada Potencial de Migração (PM) responde por "Quais" processos são candidados a migração. Cada processo BSP i computa n funções $PM(i, j)$, sendo n o número de Conjuntos e j um Conjunto. Esta abordagem tira proveito da hierarquia, de forma a não fazer todos os testes entre processos e processadores. O $PM(i, j)$ é obtido através da combinação das métricas Computação, Comunicação e Memória. A relação entre elas está baseada na noção de forças da Física. Enquanto as métricas Comunicação e Computação agem a favor da migração, a Memória age no sentido oposto. Cada métrica pode ser vista como um vetor de certa magnitude e $PM(i, j)$ como a força resultante que indica o quão apto o processo i está a migrar para o Conjunto j. A composição de cada métrica pode ser vista em detalhes em [Righi et al. 2008].

A métrica Computação - $Comp(i, j)$ - considera a predição do tempo de computação do processo i, o Padrão de Computação de i e o nível de desempenho do

Conjunto j. A predição é baseada no conceito de Aging [Tanenbaum 2003] e se baseia em informações entre duas chamadas de re-escalonamento. O Padrão de Computação mede a regularidade do processo conforme o número de instruções executadas a cada superetapa, sendo próximo a 1 caso haja constância e perto de 0 caso contrário. A média de desempenho de cada Conjunto também está inclusa na métrica, a qual é verificada localmente em cada Gerente e trocada entre eles.

Como a métrica anterior, a métrica Comunicação - $Comm(i, j)$ - calcula o Padrão de Comunicação entre processos e Conjuntos. Esse padrão considera o dados recebidos por i provenientes de processos do Conjunto j a cada superetapa. Ainda, a métrica considera a predição do tempo de comunicação entre duas chamadas de re-escalonamento. A métrica Memória - $Mem(i, j)$ - é composta pela memória do processo, a taxa de transferência entre i e o Gerente do Conjunto j e os custos de migração. Tais custos dependem do sistema operacional assim como da ferramenta que migra os processos.

$$PM(i, j) = Comp(i, j) + Comm(i, j) - Mem(i, j) \quad (3)$$

O $PM(i, j)$ é utilizado para selecionar os processos candidatos à migração e é encontrado através da Equação 3. Quanto maior o seu valor, mais propício estará o processo a migrar. Um grande $PM(i, j)$ indica que o processo é lento em sua computação, comunica-se muito com processos do Conjunto j e possui baixos custos de migração para tal Conjunto. Cada processo calcula seu PM localmente. A cada chamada de re-escalonamento, os processos passam seu maior PM ao seu Gerente. Após isso, cada Gerente troca informações com outros Gerentes. Os processos são escolhidos conforme uma lista de potenciais em ordem decrescente. Pode-se escolher o processo com maior PM ou os processos com potencial maior que $MAX(PM) \times x$, sendo x uma porcentagem.

2.4. Análise do Destino dos Processos

Cada processo i escolhido como candidato possui seu maior $PM(i, j)$ associado a um Conjunto j. Assim, o Gerente de tal Conjunto escolherá o seu processador mais adequado para receber o processo i, respondendo a questão "Onde". Antes da realização da migração, a sua viabilidade é verificada. Essa verificação se baseia em: (I) a carga externa nos processadores origem e destino; (II) os processos BSP que ambos processadores estão executando; (III) a simulação da execução do processo no processador destino; (IV) o tempo de comunicação entre os processadores; e (V) o custo da migração do processo. Com isso, calcula-se dois tempos, representando a execução local do processo (t_o) e a execução no processador destino (t_d). Por fim, o processo i é migrado (se $t_o > t_d$) ou sua migração é cancelada (se $t_o \leq t_d$). Vale lembrar que modelo não trata como os processos são migrados.

3. Metodologia de Avaliação

Esta avaliação objetiva observar o impacto do modelo MigBSP sobre aplicações BSP regulares e irregulares. A idéia aqui é verificar o ganho ou perda de desempenho obtido com migrações em ambas aplicações. Nesse contexto, foram simulados três cenários de execução: (I) aplicação; (II) aplicação com MigBSP mas migrações proibidas; e (III) aplicação com MigBSP e migrações permitidas. A comparação entre os cenários I e II mostra o impacto do modelo quando migrações são impossibilitadas devido a grandes custos de migração no ambiente. Já a análise dos cenário I e III busca apresentar o ganho ou perda de desempenho quando o modelo MigBSP é aplicado por completo no sistema.

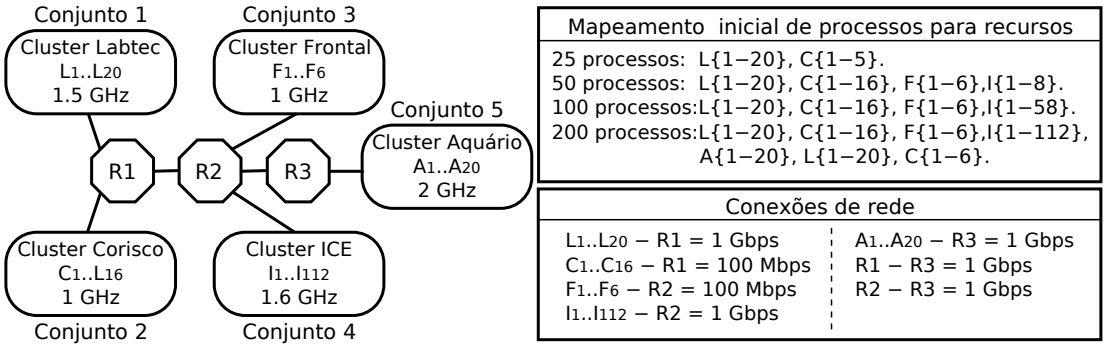

Figura 2. Ambiente simulado com 5 Conjuntos.

Para tais experimentos foi utilizado o simulador SimGrid [Casanova et al. 2008] (módulo MSG), o qual possibilita a modelagem do ambiente e aplicação, além da migração de processos. Tal simulador é determinístico. Foi montado um ambiente com 5 Conjuntos, ilustrado na Figura 2. Cada nó possui um único processador. O ambiente modela a estrutura existente na Universidade Federal do Rio Grande do Sul. Os *clusters* Labtec, Corisco e Frontal tem seus nós ligados através de Fast Ethernet, enquanto os demais *clusters* possuem Gigabit Ethernet. Os custos de migração são baseados no uso da biblioteca AMPI [Huang et al. 2006] nos *clusters*. Além do ambiente, a Figura 2 também apresenta o mapeamento inicial de processos para recursos. A idéia aqui é ocupar todos nós de um *cluster* e então seguir para o próximo, em uma espécie de escalonamento Round-robin. Foram executadas simulações com 25, 50, 100 e 200 processos.

Por fim, os testes foram executados com α *inicial* $= \{4,8,16\}$. Ainda, foram empregados $\omega = 3$ e D *inicial* $= 0,5$. Os testes com a aplicação regular consideram como candidato à migração apenas o processo com maior PM. Já com a aplicação irregular, os candidatos se encontravam entre os processos com potencial maior que $MAX(PM) \times 0,8$. A notação (p_i, P_j) indica que o processo p_i (seguindo a ordem do mapeamento inicial) migrou para o processador P_j. O ambiente simulado prioriza a questão da heterogeneidade. Trabalhos futuros incluirão testes com o modelo sobre ambientes dinâmicos através do simulador SimGrid. Esse simulador permite descrever ambientes com variações na taxa de banda passante, de latência e de capacidade de processamento.

4. Avaliação da Aplicação Regular: Método de Lattice Boltzmann

O método Lattice Boltzmann [Schepke 2007] foi escolhido como aplicação regular. Ele foi modelado no SimGrid com decomposição vertical de domínios. Todos processos BSP executam as mesmas quantidades de computação e realizam comunicação com um mesmo padrão. A cada superetapa cálculos são feitos em paralelo, seguidos da comunicação dos seus resultados a seu vizinho da direita. Esse esquema engloba diversas computações científicas, como processamento de sinais e dinâmica computacional de fluídos. Uma visão abstrata do problema envolve o algoritmo iterativo que opera sobre uma matriz retangular. A matriz é dividida em partes através de uma decomposição vertical. Cada parte é alocada para um dos processos. A cada superetapa, cada parte é atualizada localmente e informações de borda são trocadas com o processo vizinho a direita.

A matriz considerada anteriormente requer a computação de 10^{10} instruções e ocupa 10 MB de memória. Os valores usados são baseados na execução da aplicação real em *clusters* na UFRGS. A partição da matriz influencia a quantidade de instruções

a serem executadas em cada processo e no espaço em memória de cada um. Porém, o tamanho das mensagens continua a mesma. Quando são utilizados 25 processos, tem-se que cada um deles é responsável por 4×10^9 instruções, ocupa 0.9 MB (0.5MB são dados do processo) e comunica 0.1 MB com seu vizinho da direita. Ao aumentar o número de processos, diminui-se o número de instruções que cada um computa e também a região de memória que cada um usa para guardar os dados. Em outras palavras, quanto maior o número de processos, menores são as cargas de computação e custos de migração.

4.1. Análise Geral

Os resultados com a execução de 25 processos é apresentada na Tabela 1. O sistema apresentou-se estável e o valor de α cresceu a cada chamada de re-escalonamento. Considerando $\alpha = 4$, 3 migrações aconteceram na execução de 50 superetapas, cada qual em uma das três primeiras chamadas de re-escalonamento: $\{(p_{21},A_1),(p_{22},A_2),(p_{23},A_3)\}$. Uma vez que α sempre cresce, pode-se verificar que essa última migração ocorre na superetapa 1020. A partir desse ponto, não se tem mais chamadas de re-escalonamento. Além disso, com 2000 superetapas, outras 5 migrações ocorreram: $\{(p_{24},A_4),(p_{25},A_5)$, $(p_{18},A_6),(p_{19},A_7),(p_{20},A_8)\}$. O que se percebe aqui é que todas as migrações aconteceram para o *cluster* mais rápido - Aquário. Ainda, as primeiras 5 migrações moveram processos do *cluster* Corisco e as 3 restantes trataram de processos localizados previamente no *cluster* Labtec. Por fim, obteve-se um ganho de 15% após a execução de 2000 superetapas, comparando os cenários I e III.

Tabela 1. Avaliação de 25 processos nos 3 cenários (tempo em segundos).

Superetapa	Cenário I	alfa = 4		alfa = 8		alfa = 16	
		Cenário II	Cenário III	Cenário II	Cenário III	Cenário II	Cenário III
50	17,35	19,32	20,45	18,66	19,44	18,66	19,42
500	173,53	177,46	154,87	176,80	161,48	176,80	179,24
2000	694,12	699,47	592,26	698,68	599,14	697,43	613,88

Analisando os resultados do cenário III com $\alpha = 16$, foi detectado que a primeira migração acaba sendo adiada, o que resulta em um tempo de execução maior se comparado a valores menores de α. Como pode ser observado na Tabela 1, com $\alpha = 4$ tem-se um maior impacto na execução de 50 superetapas devido ao maior número de chamadas de re-escalonamento no início da aplicação. Entretanto, esse sobrecusto é amortizado com o aumento no número de superetapas. Uma vez que as migrações acontecem mais rapidamente com $\alpha = 4$, observa-se que o tempo das superetapas é reduzido, mas o custo de cada migração acaba adicionado ao tempo total de execução da aplicação.

Tabela 2. Avaliação de 50 processos nos 3 cenários (tempo em segundos).

Superetapa	Cenário I	alfa = 4		alfa = 8		alfa = 16	
		Cenário II	Cenário III	Cenário II	Cenário III	Cenário II	Cenário III
50	10,78	13,14	14,47	12,35	13,32	12,35	13,03
500	107,74	112,46	106,90	111,67	115,73	111,67	117,84
2000	430,95	436,79	408,25	435,88	417,56	434,68	434,30

A Tabela 2 exibe os resultados obtidos com 50 processos. O sistema encontrou-se estável, pois os processos estavam balanceados entre os recursos utilizados. O mesmo esquema de migrações dos testes com 25 processos aconteceu com a

configuração de 50 processos. Porém, se obteve a migração de 8 diferentes processos ao considerarmos 2000 superetapas e $\alpha = 4$: $\{(p_{38},A_1),(p_{40},A_2),(p_{42},A_3),(p_{39},A_4),(p_{41},A_5),(p_{37},A_6),(p_{22},A_7),(p_{21},a_8)\}$. MigBSP move todos processos do *cluster* Frontal e 2 do *cluster* Corisco para o *cluster* Aquário (o mais rápido). Considerando $\alpha = 4$, obteve-se um ganho de 5% com o emprego de MigBSP. Além disso, ganhos também ocorreram com $\alpha = 8$. Por outro lado, o resultado final com $\alpha = 16$ no cenário III é pior do que aquele do cenário I, porque as migrações são atrasadas. Nesse contexto, são necessárias mais superetapas para se obter algum ganho com a migração de processos

Tabela 3. Avaliação de 100 processos nos 3 cenários (tempo em segundos).

Superetapa	Cenário I	alfa = 4		alfa = 8		alfa = 16	
		Cenário II	Cenário III	Cenário II	Cenário III	Cenário II	Cenário III
50	5,94	8,59	9,63	7,71	8,48	7,71	8,19
500	59,25	64,57	62,55	63,68	67,25	63,68	69,37
2000	236,96	243,70	224,48	241,12	232,87	239,23	241,52

Os resultados com a execução de 100 processos é apresentada na Tabela 3. Como nos experimentos com 25 e 50 processos, o sistema encontrou-se estável. As mesmas migrações aconteceram nos testes com 50 e 100 processos, pois com essa última quantia apenas ocupa-se um número maior de nós no *cluster* ICE. A Tabela 4 apresenta os resultados obtidos com 200 processos. Neste caso, obteve-se um sistema instável (com desbalanceamento de carga). Isso faz com que se tenha um maior sobrecusto no cenário II. Ainda nesse cenário, o valor de α começou a crescer após ω chamadas de re-escalonamento sem migrações (segundo o Algoritmo 2 na Subseção 2.2.2). Considerando o cenário III e $\alpha = 4$, 12 migrações aconteceram nas primeiras 50 superetapas: $\{(p_{195},A_1),(p_{197},A_2),(p_{196},A_3),(p_{198},A_4),(p_{199},A_5),(p_{200},A_6),(p_{38},A_7),(p_{39},A_8),(p_{37},A_9),(p_{40},A_{10}),(p_{41},A_{11}),(p_{42},A_{12})\}$. Mesmo assim o sistema continua desbalanceado. MigBSP indica que não há viabilidade para migrações nas outras chamadas de re-escalonamento. Portanto, após ω chamadas sem migrações, o valor de D é aumentado (segundo à Adaptação 2), o que leva ao aumento de α. Após esse procedimento, o sistema passa a ser considerado estável e α cresce a cada chamada de re-escalonamento. Entretanto, a localização dos processos continuou a mesma.

Tabela 4. Avaliação de 200 processos nos 3 cenários (tempo em segundos).

Superetapa	Cenário I	alfa = 4		alfa = 8		alfa = 16	
		Cenário II	Cenário III	Cenário II	Cenário III	Cenário II	Cenário III
50	5,09	10,56	17,14	9,65	11,06	7,82	8,15
500	50,66	57,84	68,44	56,93	71,42	55,92	77,05
2000	200,43	209,46	194,87	208,13	202,22	204,69	211,69

Por fim, observou-se que quanto maior o número de processos, melhor o desempenho obtido no cenário I. Porém, sabe-se que há um limite para tal, a partir do qual o aumento de processos acaba por degradar o desempenho. O modelo MigBSP obteve melhores resultados nos experimentos com 25 processos. Com o aumento no número de processos tem-se uma quantidade de computação menor por processo e um aumento da comunicação no sistema. Assim, precisa-se de um maior número de superetapas para se obter um ganho no desempenho através das migrações. A Figura 3 ilustra o sobrecusto do modelo quando não se tem migrações. Conclui-se que ambas as adaptações tornam o modelo viável e leve no tratamento do re-escalonamento de processos BSP.

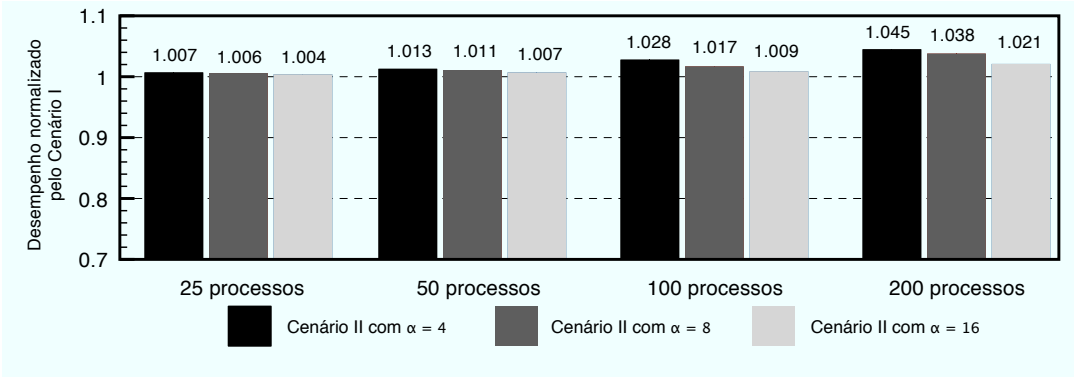

Figura 3. Desempenho no cenário II com diferentes números de processos.

4.2. Impacto no Tempo da Barreira de Sincronização

A Tabela 5 apresenta o tempo da barreira de sincronização na superetapa 2000. A sincronização foi organizada de forma centralizada numa abordagem mestre-escravo, onde p_1 troca mensagens com todos processos BSP, sendo o tempo da barreira capturado nele. Ao analisar a Tabela 5, pode-se concluir que MigBSP reduz os custos de comunicação e, consequentemente, o tempo da barreira de sincronização. Essa tabela informa um ganho de 22% no tempo obtido com o modelo na execução de 50 processos. Tal ganho foi reduzido nos experimentos com 100 processos, devido ao fato que apenas se tem mais processos incluídos no *cluster* ICE.

Tabela 5. Tempos das barreiras (em milissegundos) em diferentes cenários.

Processos	25	50	100	200
Cenário I	23,94	33,49	36,13	43,25
Cenário III	10,77	25,36	28,34	31,44

5. Avaliação da Aplicação Irregular: Algoritmo de Smith-Waterman

A aplicação BSP irregular simulada é baseada em Programação Dinâmica (PD). PD é um método de resolução de problemas que apresentam subestruturas ótimas com superposição [Low et al. 2007]. Algoritmos desenvolvidos com tal método podem ser classificados de acordo com o tamanho da matriz e das dependências entre cada célula da matriz. Um algoritmo para um problema de tamanho n é chamado algoritmo tD/eD se o tamanho da matriz é $O(n^t)$ e cada célula depende de outras $O(n^e)$ células. O algoritmo de Smith-Waterman foi escolhido para o experimento por ser um algoritmo $2D/1D$ bem conhecido (usado para sequenciamento de DNA). Tal algoritmo apresenta irregularidade na carga computacional entre as células da matriz.

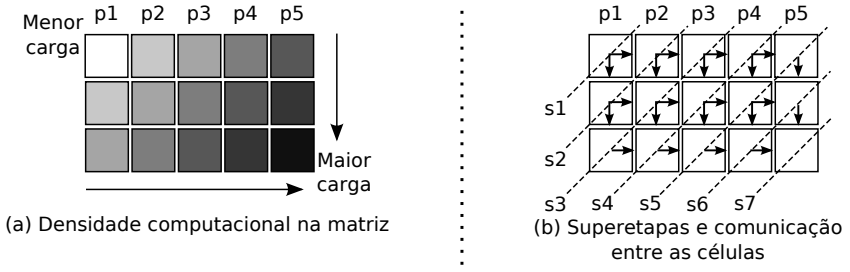

Figura 4. Aplicação BSP irregular: algoritmo de Smith Waterman.

Como todo algoritmo BSP, Smith-Waterman apresenta uma computação em fases. Esse algoritmo segue em uma série de ondas diagonais através da matriz. Considerando as dependências entre as células, somente células em uma mesma antidiagonal podem ser executadas em paralelo. A Figura 4 (a) ilustra o conceito do algoritmo em uma matriz 3×5 com um mapeamento de colunas para os processos. Quanto mais escura a célula, maior a sua carga computacional. Cada onda corresponde a uma superetapa. Por exemplo, a Figura 4 (b) apresenta as dependências entre as células de uma matriz que executa em 7 superetapas. Essa organização mostra que para uma matriz de dimensões $l\times c$ serão executadas $l+c-1$ superetapas. Além disso, cada processo estará envolvido em l superetapas. Uma vez que foi empregado o mapeamento de colunas para os processos, tem-se somente comunicações de um processo para seu vizinho da direita.

A configuração dos cenários II e III depende do Padrão de Computação $P_{comp(i)}$ de cada processo i. O valor de $P_{comp(i)}$ aumenta ou diminui conforme a predição da quantidade de instruções executadas em cada superetapa. $PI_t(i)$ representa tal predição em uma superetapa t. Ela é baseada no conceito de Aging [Tanenbaum 2003] que faz com que os valores mais recentes tenham maior influência na predição. A forma genérica do cálculo de $PI_t(i)$ é apresentada abaixo. $I_t(i)$ representa a quantidade de instruções executadas pelo processo i na superetapa t. Um processo é considerado regular se a previsão se encontra dentro de uma margem δ. O Algoritmo 3 descreve a computação de $P_{comp(i)}$ entre duas chamadas para re-escalonamento. $P_{comp(i)}$ recebe o valor 1 no início da aplicação, uma vez que é feita uma suposição inicial que todos os processos são regulares quanto as suas computações no decorrer das superetapas.

$$PI_t(i) = \begin{cases} I_t(i) & se\ t = k \\ \frac{1}{2}PI_{t-1}(i) + \frac{1}{2}I_t(i) & se\ k < t \leq k+\alpha-1 \end{cases}$$

Algoritmo 3 Padrão de Computação $P_{comp}(i)$

1: **para** t da superetapa k até a superetapa $k+\alpha-1$ **faça**
2: **se** $PI_t(i) \geq I_t(i).(1-\delta)$ e $PI_t(i) \leq I_t(i).(1+\delta)$ **então**
3: Aumenta $P_{comp}(i)$ de $\frac{1}{\alpha}$ com um máximo de 1
4: **senão**
5: Diminui $P_{comp}(i)$ de $\frac{1}{\alpha}$ com um mínimo de 0
6: **fim**
7: **fim**

Nos experimentos foram utilizados 10^6 como a quantidade de instruções na primeira superetapa e 10^9 na última. O crescimento na carga computacional entre as superetapas é uniforme. Assim, foi utilizado um valor de δ igual a $0,01$ para o cenário II e $0,50$ para o cenário III. O segundo caso foi escolhido dessa forma pois induz uma regularidade no sistema considerando o número de instruções realizadas pelo processos a cada superetapa. Ambos valores de δ influenciam a métrica Computação e, consequentemente, a escolha dos candidatos à migração. No cenário II obtém-se valores negativos para o PM, impedindo as migrações. Isso acontece devido ao valor da métrica Computação ser próximo a 0. Os experimentos envolvem o mapeamento de colunas para processos (como ilustrado na Figura 4 (b)). Cada processo possui 0.7 MB de memória fixa. Além disso, 5 MB de dados são divididos entre os processos. Cada processo envia mensagens de tamanho igual a sua quantidade de dados.

5.1. Análise Geral

A Tabela 6 apresenta os resultados obtidos com a simulação dos 3 cenários, considerando os diferentes tamanhos de matriz. Os resultados gerais para os cenários II e III podem ser vistos normalizados pelo cenário I nas Figuras 5 e 6, respectivamente. O sistema apresentou-se estável nos experimentos com a matriz 25×25, a qual executou 49 superetapas. Os testes com α = 8 apresentaram um sobrecusto no cenário III quando comparado ao cenário I. Duas chamadas de re-escalonamento foram feitas, nas superetapas 8 e 24. A primeira causou apenas 1 migração $\{(p_1,A_1)\}$. Já a segunda resultou em 3 migrações: $\{(p_{21},A_2),(p_{22},A_3),(p_{23},A_4)\}$. Observou-se que os processos p_{24} e p_{25} continuaram no *cluster* Corisco, o que comprometeu as superetapas em que executaram, pois a barreira sempre espera pelo processo mais lento. Considerando α = 16, chamadas de re-escalonamento foram feitas nas superetapas 16 e 48. Apesar da última chamada ter migrado processos lentos para o *cluster* Aquário, a aplicação executou apenas mais 1 superetapa. Vale lembrar que o MigBSP trabalha sem conhecimento prévio da aplicação.

Tabela 6. Avaliação dos 3 cenários variando o tamanho da matriz (tempo em segundos).

Matrizes	Cenário I	Cenário II			Cenário III		
		alfa = 4	alfa = 8	alfa = 16	alfa = 4	alfa = 8	alfa = 16
25x25	40,74	42,24	41,63	41,28	34,82	41,64	40,64
50x50	92,59	94,84	94,03	93,36	84,65	83,00	85,21
100x100	162,66	165,66	164,80	164,04	148,89	146,55	162,49
200x200	389,91	393,68	392,85	392,01	375,53	374,38	374,40

Os resultados obtidos com a matriz 50×50 são apresentados na Tabela 6. A aplicação executou 99 superetapas. O sistema apresentou-se estável, com um valor de α crescente. Todos valores de α apresentaram ganhos no cenário III. Com α = 4, obteve-se 84,65 segundos para o cenário III, o que representa um ganho de 9%. Considerando α = 8, 3 chamadas de re-escalonamento aconteceram: nas superetapas 8, 24 e 56. Somente as duas últimas apresentaram migrações. Isso aconteceu devido ao fato dos custos de migração serem maiores que as quantidades de computação e comunicação no início da aplicação. Três processos foram migrados na superetapa 24: $\{(p_{21},A_1),(p_{22},A_2),(p_{23},A_3)\}$. Na superetapa 56, os processos de 37 a 42 foram migrados para o *cluster* Aquário. Esse re-escalonamento mostrou-se eficiente pois transferiu todos os processos do *cluster* Frontal para Aquário. Já com α = 16, foram executadas 2 chamadas de re-escalonamento. Na superetapa 16, os processo de ordem 1 a 15 foram migrados para o *cluster* Aquário. Na superetapa 48, 6 processos foram escolhidos como candidatos para migração, mas apenas 5 migraram devido a ocupação do *cluster* Aquário.

A execução de 100 processos (matriz 100×100 e 199 superetapas) apresentou bons resultados com migrações, como pode ser visto na Figura 6. Considerando α = 16, 3 chamadas de re-escalonamento foram feitas, todas apresentando migrações. Na superetapa 16, os primeiros 11 processos foram migrados para o *cluster* Aquário. Todos processos do *cluster* Frontal foram migrados para o *cluster* Aquário na superetapa 48. Isso acontece pois o MigBSP dá preferência à migração dos processos alocados nos *clusters* mais lentos. Por fim, os processos do *cluster* Corisco ($p_{21..35}$) foram selecionados para a migração na superetapa 112. Porém, apenas 3 deles foram migrados, pois somente 3 processadores do *cluster* Aquário estavam ainda vagos.

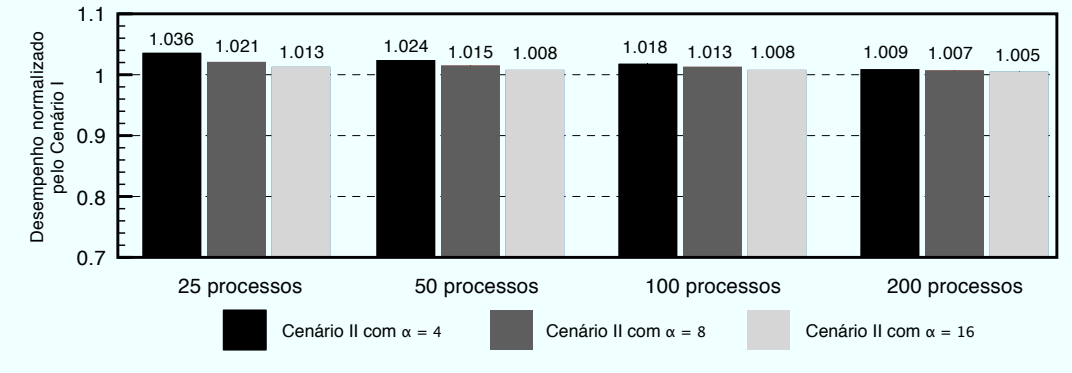

Figura 5. Desempenho no cenário II com diferentes números de processos.

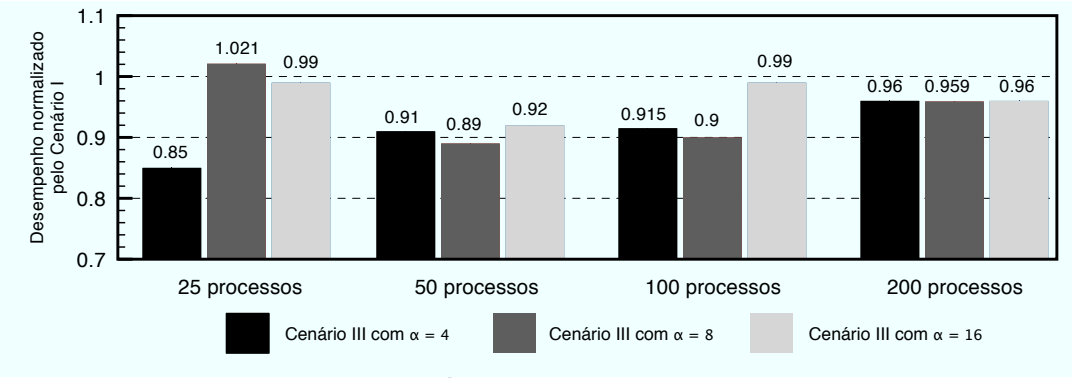

Figura 6. Desempenho no cenário III com diferentes números de processos.

O sobrecusto obtido no cenário II comparado ao cenário I na simulação da matriz 200×200 apresentou-se menor que 1%, como ilustrado na Figura 5. Em adição, resultados satisfatórios foram obtidos com a migração de processos. Mesmo tendo mais de um processo mapeado para um mesmo processador, a maior parte das vezes apenas um dos processos apresenta-se ativo em determinada superetapa. Além disso, o grande número de superetapas (399) ajuda a diluir o sobrecusto do modelo. Os resultados obtidos no cenário III, como pode ser visto na Figura 6, foram muito similares. Isso ocorre parcialmente devido as últimas chamadas de re-escalonamento acontecerem próximas ao meio da execução da aplicação para todos os valores de α testados.

6. Trabalhos Relacionados

Usualmente, componentes específicos e *frameworks* são utilizados para prover migração. Relativo a isso, Bhandarkar, Brunner e Kale apresentam o suporte para o balanceamento de carga adaptativo em aplicações MPI [Bhandarkar et al. 2000]. Periodicamente, a aplicação transfere o controle ao balanceador de carga através da função $MPI\_Migrate()$. Estes autores apresentam um estratégia que utiliza do grafo de comunicação para re-escalonar os processos. Adaptive MPI (AMPI) [Huang et al. 2006] usa o *framework* Charm++ para oferecer balanceamento de carga automático. Charm++ coleta informações sobre a carga de trabalho e padrões de comunicação e as utiliza nos momentos de re-escalonamento. O sistema GrADS [Vadhiyar and Dongarra 2005] apresenta suporte a migrações e adaptatividade. Entretanto, o custo de migração é considerado como um valor fixo e os ganhos com o re-escalonamento são baseados no tempo restante de execução. Assim, este *framework* necessita de aplicações com partes conhecidas previamente.

Du, Sun e Wu [Du et al. 2007] descreveram um modelo que considera os estados dos processos, memória, E/S e comunicação para medir os custos de migração durante a execução de uma aplicação. Porém, os autores não especificam quando re-escalonar os processos, nem quais devem ser migrados. Kondo et al. [Kondo et al. 2002] descreveram um modelo de escalonamento cliente-servidor para computações em escala global. O modelo considera a velocidade do processador, a largura de banda da rede e o espaço em disco para determinar a quantidade de tarefas que um cliente pode receber. Entretanto, esses valores não são combinados, considerando-se apenas o menor deles para definir a quantidade de trabalho a ser enviada a um cliente.

Considerando o modelo de programação BSP, podem ser citados dois trabalhos que apresentam migrações. O primeiro descreve a biblioteca PUBWCL, a qual busca aproveitar ciclos ociosos de nós espalhados pela Internet [Bonorden et al. 2005]. A biblioteca oferece migração a cada superetapa (durante a fase de computação ou após a barreira). Todos algoritmos propostos consideram apenas os tempos de computação. O segundo trabalho trata uma extensão da biblioteca PUB [Bonorden 2007] para o suporte à migrações. O autor explica que o balanceador de carga decide quando lançar a migração de processos, mas o assunto não é tratado no trabalho. As estratégias propostas pelo autor não consideram a comunicação entre os processos nem os custos de migração.

7. Conclusões e Trabalhos Futuros

O escalonamento para sistemas paralelos pode ser visto em dois níveis [Frachtenberg and Schwiegelshohn 2008]. No primeiro nível, processadores são alocados para um trabalho. No segundo nível, processos de um trabalho são (re-)escalonados utilizando um conjunto de processadores. MigBSP pode ser incluído no segundo nível, oferecendo algoritmos para rebalanceamento de carga (processos) entre recursos durante a execução da aplicação. Considerando a migração em aplicações BSP, MigBSP oferece as seguintes contribuições: (I) combinação de múltiplas métricas para medir o potencial de migração de cada processo; (II) consideração de padrões para analisar a regularidade de cada processo; e (III) adaptatividade no re-escalonamento de acordo com o estado do sistema.

O modelo considera informações sobre os processos e recursos para gerar a métrica *PM*. Tal métrica trata do poder de migração de um processo i a um Conjunto j. Assim, nem todos testes possíveis são feitos no momento do re-escalonamento. Contrário a biblioteca PUBWCL [Bonorden et al. 2005], MigBSP utiliza-se de adaptações na frequência das chamadas de re-escalonamento para reduzir o seu impacto no tempo de execução da aplicação. As principais idéias aqui envolvem o atraso da chamada de re-escalonamento caso o sistema esteja balanceado e/ou caso um padrão de chamadas sem migrações se apresente por múltiplas tentativas. Referente a segunda contribuição, fica estabelecido que a regularidade dos processos é importante para MigBSP, pois espera-se que eles apresentem comportamentos similares nos seus processadores destino.

Considerando ambas sessões de avaliação, conclui-se que o modelo MigBSP obteve bons resultados com as aplicações regular e irregular testadas. Inicialmente, o modelo foi desenvolvido para tratar com aplicações regulares. Entretanto, ele dispõe de parâmetros que tornam possível o ajuste de sua funcionalidade para tratar com a irregularidade de forma eficiente. Observou-se que o ganho com as migrações está fortemente

ligado a quantidade de computação feita em cada processo. No escopo da aplicação irregular, quanto maior a carga computacional nas últimas superetapas, melhor serão os resultados com as migrações dos últimos processos. Ainda, em situações em que se tem uma baixa razão entre computação e comunicação, são necessários uma maior quantidade de superetapas para se obter um ganho com a migração. Por fim, MigBSP pode ser apresentado como um modelo viável para re-escalonamento de processos BSP, obtendo ganhos maiores que 15% e sobrecustos menores que 5% quando migrações não são possíveis. É importante reforçar que tal abordagem **exclui a necessidade de execuções prévias dos algoritmos e não necessita de conhecimento da aplicação**, somente necessitando ligar a aplicação ao *middleware* de re-escalonamento.

Além da avaliação da dinamicidade no ambiente, trabalhos futuros incluem a simulação do modelo MigBSP junto ao Grid5000. A intenção desse teste é verificar a escalabilidade do modelo sobre tal plataforma. Ainda, pretende-se analisar o autoajuste dos pesos das métricas baseado no trabalho de Wieczorek et al. [Wieczorek et al. 2008].

Referências

Aggarwal, G., Motwani, R., and Zhu, A. (2003). The load rebalancing problem. In *SPAA '03: Proceedings of the fifteenth annual ACM symposium on Parallel algorithms and architectures*, pages 258–265, New York, NY, USA. ACM Press.

Bhandarkar, M. A., Brunner, R., and Kale, L. V. (2000). Run-time support for adaptive load balancing. In *IPDPS '00: Proceedings of the 15 IPDPS 2000 Workshops on Parallel and Distributed Processing*, pages 1152–1159, London, UK. Springer-Verlag.

Bonorden, O. (2007). 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.

Bonorden, O., Gehweiler, J., and auf der Heide, F. M. (2005). Load balancing strategies in a web computing environment. In *Proceeedings of International Conference on Parallel Processing and Applied Mathematics (PPAM)*, pages 839–846, Poznan, Poland.

Casanova, H., Legrand, A., and Quinson, M. (2008). 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. IEEE Computer Society.

Casavant, T. L. and Kuhl, J. G. (1988). A taxonomy of scheduling in general-purpose distributed computing systems. *IEEE Trans. Softw. Eng.*, 14(2):141–154.

Chen, C. and Tong, W. (2004). The application of the bsp model on datagrid. In *SCC '04: Proceedings of the 2004 IEEE International Conference on Services Computing*, pages 471–474, Washington, DC, USA. IEEE Computer Society.

Du, C., Sun, X.-H., and Wu, M. (2007). Dynamic scheduling with process migration. In *CCGRID '07: Proceedings of the Seventh IEEE International Symposium on Cluster Computing and the Grid*, pages 92–99, Washington, DC, USA. IEEE Computer Society.

Frachtenberg, E. and Schwiegelshohn, U. (2008). New Challenges of Parallel Job Scheduling. *Job Scheduling Strategies for Parallel Processing*, (4942):1–23.

García, E. W. and Morales-Luna, G. (2008). Simulation for bulk synchronous parallel superstep task assignment in desktop grids characterised by gaussian parameter distributions. *Multiagent and Grid Systems*, 4(2):141–166.

Goldchleger, A., Kon, F., Goldman, A., Finger, M., and Bezerra, G. C. (2004). Integrade object-oriented grid middleware leveraging the idle computing power of desktop machines: Research articles. *Concurr. Comput. : Pract. Exper.*, 16(5):449–459.

Huang, C., Zheng, G., Kale, L., and Kumar, S. (2006). Performance evaluation of adaptive mpi. In *PPoPP '06: Proceedings of the eleventh ACM SIGPLAN symposium on Principles and practice of parallel programming*, pages 12–21, New York, NY, USA. ACM Press.

Kondo, D., Casanova, H., Wing, E., and Berman, F. (2002). Models and scheduling mechanisms for global computing applications. In *IPDPS '02: Proceedings of the 16th International Symposium on Parallel and Distributed Processing*, page 79.2, Washington, DC, USA. IEEE Computer Society.

Kovacs, J. and Kacsuk, P. (2004). A migration framework for executing parallel programs in the grid. In *European Across Grids Conference*, volume 3165 of *Lecture Notes in Computer Science*, pages 80–89. Springer.

Low, M. Y.-H., Liu, W., and Schmidt, B. (2007). 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.

Righi, R. d. R., Pilla, L. a. L., Carissimi, A., and Navaux, P. O. (2008). Controlling processes reassignment in bsp applications. In *Computer Architecture and High Performance Computing, 2008. SBAC-PAD '08. 20th International Symposium on*, pages 37–44.

Schepke, Claudio; Maillard, N. (2007). Performance improvement of the parallel lattice boltzmann method through blocked data distributions. In *19th International Symposium on Computer Architecture and High Performance Computing, 2007. SBAC-PAD 2007*, pages 71–78.

Skillicorn, D. B., Hill, J. M. D., and McColl, W. F. (1997). Questions and Answers about BSP. *Scientific Programming*, 6(3):249–274.

Tanenbaum, A. (2003). *Computer Networks*. Prentice Hall PTR, Upper Saddle River, New Jersey, 4th edition.

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

Valiant, L. G. (1990). A bridging model for parallel computation. *Commun. ACM*, 33(8):103–111.

Wieczorek, M., Podlipnig, S., Prodan, R., and Fahringer, T. (2008). Bi-criteria scheduling of scientific workflows for the grid. *International Symposium on Cluster Computing and the Grid (CCGrid)*, pages 9–16.