Auto-Organização na Seleção de Candidatos a Migração em Aplicações Bulk Synchronous Parallel

Lucas Graebin[1], **Diogo Pinheiro de Araújo**[1], **Rodrigo da Rosa Righi**[1]

[1]Programa Interdisciplinar de Pós-Graduação em Computação Aplicada
Universidade do Vale do Rio dos Sinos
CEP: 93.022-000 – São Leopoldo – RS – Brasil

lgraebin@acm.org, {dapinheiro,rrrighi}@unisinos.br

Abstract. *Process rescheduling can be a suitable technique for achieving high performance, mainly when irregular workloads and cluster-of-clusters environments are considered. The decision to move processes to new resources is NP-Hard, and heuristics take place in order to reach good results inside an acceptable time interval. In this way, this paper presents AutoMig – a novel heuristic for BSP (Bulk Synchronous Parallel) applications that self-organizes the selection of candidates for migration on different clusters. Its differential approach consists in a prediction function (pf) that considers processes' computation and communication data as well as their migrations costs. pf is applied over a list of schedules and AutoMig's final step decides whether one of them outperforms the current mapping. The results emphasize gains up to 32% when testing a CPU-bound application in a simulated cluster-of-clusters environment.*

Resumo. *Reescalonamento de processos pode ser uma técnica pertinente para atingir melhor desempenho, principalmente quando cargas irregulares e ambientes com múltiplos clusters são considerados. A decisão de mover processos para novos recursos é NP-Difícil, e heurísticas são usadas com o intuito de atingir bons resultados em um tempo aceitável. Nesse sentido, esse artigo apresenta AutoMig – uma nova heurística para aplicações BSP (Bulk Synchronous Parallel) que auto-organiza a seleção dos candidatos a migração para diferentes clusters. Seu diferencial é uma função de predição (pf) que considera dados de computação e comunicação dos processos, bem como os custos de migração. pf é aplicada sobre uma lista de escalonamentos e AutoMig decide qual é o mais adequado deles para atingir um desempenho melhor que o oferecido pelo mapeamento corrente. Os resultados mostram ganhos de até 32% nos testes de uma aplicação CPU-Bound em um ambiente simulado com múltiplos clusters.*

1. Introdução

Na medida em que a computação distribuída é cada vez mais explorada, recursos de múltiplos *clusters* se tornam os nós de computação dominantes na formação de *grids* [El Kabbany et al. 2011, Qin et al. 2010, Xhafa and Abraham 2010]. Aplicações contendo rotinas para a resolução de sistemas lineares e Transformada Rápida de Fourier (FFT) são exemplos típicos de aplicações paralelas fortemente acopladas que podem tirar proveito de processamento de múltiplos *clusters* [Sanjay and Vadhiyar 2009]. Uma vez que esses *clusters* podem ser heterogêneos entre si e que ligações inter-clusters são

normalmente mais lentas que as usadas internamente, questões como escalonamento e balanceamento de carga devem ser tratados com o objetivo de garantir bom desempenho nessa arquitetura. Uma alternativa para oferecer ambos os tratamentos é o emprego da migração de processos. Assim, é possível remodelar o mapeamento processos-recursos em tempo de execução em resposta ao comportamento dos processos e dos recursos (processadores e rede).

Geralmente, migração de processos é implementada junto com a própria aplicação paralela, resultando numa implementação específica e não extensível para um determinado problema. Além disso, há iniciativas que trabalham com chamadas explícitas dentro do código da aplicação [Bhandarkar et al. 2000] e/ou obrigam execuções extras para conseguir dados de entrada para o escalonador [Silva et al. 2005, Yang and Chou 2009]. Uma abordagem diferente para a migração acontece em nível de *middleware*, onde mudanças na aplicação e conhecimento prévio do sistema normalmente não são requeridos. Nesse sentido, foi desenvolvido um modelo de reescalonamento de processos chamado MigBSP [da Rosa Righi et al. 2010]. Tal modelo foi projetado para operar sobre aplicações baseadas em fases que seguem o estilo BSP (*Bulk Synchronous Parallel*) e atua sobre arquiteturas com múltiplos *clusters*. Além disso, MigBSP faz uso de heurísticas em seus algoritmos porque o problema de encontrar um mapeamento ótimo em sistemas heterogêneos é um problema NP-Difícil [Xhafa and Abraham 2010].

Considerando a escolha dos processos para reescalonamento, o modelo MigBSP cria uma lista de prioridade baseada no maior Potencial de Migração (PM) de cada processo [da Rosa Righi et al. 2010]. PM combina os custos de migração com dados vindos das fases de computação e comunicação para oferecer uma métrica de escalonamento unificada. O processo presente no topo da lista é sempre escolhido para ser inspecionado quanto a migração. Embora essa abordagem acarretasse em bons resultados, concorda-se que uma outra mais otimizada poderia lidar com múltiplos processos quando a chamada de reescalonamento acontece. Uma possibilidade seria selecionar uma percentagem de processos baseado no maior PM. Contudo, uma questão é pertinente: Como é possível achar uma porcentagem otimizada para aplicações dinâmicas e ambientes heterogêneos? A solução poderia envolver o teste de várias instâncias de parâmetros. Certamente, essa ideia consome tempo e novas aplicações e recursos requerem um novo conjunto de testes.

Ao longo do desenvolvimento da versão de MigBSP descrita acima, foi observado o crescimento do uso de sistemas de escalonamento os quais ajustam parâmetros em tempo de execução e escondem complexidades de configuração e decisões de otimização dos usuários [Ding et al. 2009, Nascimento et al. 2007, Sanjay and Vadhiyar 2009]. Nesse contexto, foi desenvolvida uma heurística chamada **AutoMig** que seleciona um ou mais candidatos a migração automaticamente. Ela tira proveito dos conceitos de *List Scheduling* [Duselis et al. 2009] e *Backtracking* [Baritompa et al. 2007] para avaliar, de maneira autônoma, o impacto da migração sobre cada elemento da lista de PM. Em adição, outra característica de AutoMig é a não necessidade de informar parâmetros adicionais em MigBSP para selecionar múltiplos processos para migração. A avaliação do escalonamento usa uma função de predição (*pf*) que considera os custos de migração e trabalha segundo os fundamentos de uma superetapa BSP [Bonorden 2007]. O menor valor obtido indica o melhor plano obtido para o reescalonamento dos processos.

O objetivo desse artigo é descrever AutoMig. Para isso foi construída uma

aplicação que computa a compressão de imagens baseada na técnica de Fractal [Guo et al. 2009]. Uma vez que o programador não modifica sua aplicação para usar AutoMig, considera-se os resultados cujos ganhos médios foram de 7.9% satisfatórios. Tal classificação também se deve ao fato de AutoMig não precisar de dados da aplicação e dos recursos em tempo de projeto ou compilação. Os resultados mostraram situações onde AutoMig passa o desempenho da heurística que escolhe um só processo.

O presente artigo está organizado da seguinte forma. A próxima seção mostra MigBSP e serve como base para as explicações sobre a heurística proposta. A Seção 3 descreve essa heurística. Ambas Seções 4 e 5 apresentam a metodologia e os resultados experimentais, respectivamente. Trabalhos relacionados são discutidos na Seção 6. Por fim, a Seção 7 conclui o artigo e descreve trabalhos futuros.

2. MigBSP: Modelo de Reescalonamento de Processos

MigBSP responde as seguintes questões sobre balanceamento de carga: (i) "Quando" lançar a migração; (ii) "Quais" processos são candidatos a migração; (iii) "Onde" colocar os processos eleitos. As ideias para o tratamento dessas questões estão apresentadas em detalhes no trabalho [da Rosa Righi et al. 2010]. MigBSP trabalha sobre uma arquitetura heterogênea composta por *clusters*, supercomputadores e/ou redes locais. A heterogeneidade trata de processadores com diferentes velocidades e redes com larguras de banda e níveis distintos (Fast e Gigabit Ethernet e ambientes com múltiplos *clusters*, por exemplo). Tal arquitetura é montada com Conjuntos e Gerentes de Conjuntos. Um Conjunto pode ser uma LAN ou um *cluster*. Cada Conjunto possui um Gerente de Conjunto, o qual captura informações sobre seu Conjunto e as troca com os demais Gerentes.

A decisão de remapeamento dos processos é tomada no final de cada superetapa. Com o intuito de gerar menos intrusividade na aplicação, MigBSP atua com duas adaptações que controlam o valor de α ($\alpha \in \mathbb{N}^*$). α é atualizado a cada chamada de reescalonamento e irá indicar o intervalo para o lançamento da próxima. Os objetivos das adaptações são: (i) postergar a chamada de reescalonamento se os processos estão balanceados ou a tornar mais frequente, caso contrário; (ii) atrasar a chamada de reescalonamento caso um padrão sem migrações é observado nas ω últimas chamadas. Uma variável D é usada para denotar a percentagem de quão longe o processo mais lento e o mais rápido podem estar da média para classificar os processos como balanceados. Em suma, quanto maior o valor de α, menor o impacto do modelo na execução da aplicação.

A resposta à questão "Quais" é atingida através da função de decisão chamada Potencial de Migração (PM). Cada processo i computa n funções $PM(i, j)$, onde n é o número de Conjuntos e j significa um Conjunto específico. O diferencial nessa abordagem é a realização de um subconjunto de testes processos-recursos no momento do reescalonamento. $PM(i, j)$ é obtido através da combinação das métricas Computação, Comunicação e Memória (veja Equação 1). A relação entre as métricas está baseada na noção de força da física. Computação e Comunicação são métricas que atuam a favor da migração, enquanto a Memória trabalha em direção oposta. Na medida em que o valor de $PM(i, j)$ aumenta, aumenta também a possibilidade de migração do processo avaliado.

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

A métrica Computação – $Comp(i, j)$ – considera três informações: (i) predição

do tempo de computação do processo i; (ii) Padrão de Computação do processo i; (iii) grau de desempenho do Conjunto j. A predição é baseada no conceito de Aging [Tanenbaum 2003] e usa dados capturados entre duas chamadas de reescalonamento. O Padrão de Computação mede a estabilidade, ou regularidade, do processo levando em conta a quantidade de instruções realizadas a cada superetapa. Esse valor é perto de 1 caso o processo apresente regularidade e próximo de 0 caso contrário. Assim como a métrica de Computação, a métrica Comunicação – $Comm(i, j)$ – computa o Padrão de Comunicação entre processos e Conjuntos. Esse padrão considera a quantidade de dados recebidos pelo processo i de processos que pertencem ao Conjunto j a cada superetapa. Em adição, essa métrica também usa uma predição do tempo de comunicação entre ativações de rebalanceamento. A métrica Memória – $Mem(i, j)$ – trabalha com os seguintes dados: (i) memória do processo; (ii) tempo de transferência entre o processo i e o Gerente do Conjunto j; (iii) custos de migração. Esse último tópico depende do sistema operacional bem como das ferramentas usadas para a migração de processos.

Os processos calculam $PM(i, j)$ localmente. A cada chamada para migração, eles passam o seu maior $PM(i, j)$ para seus gerentes. Esta entidade troca o PM recebido com os demais gerentes. Como já mencionado, existe uma heurística para escolher o candidato, a qual captura o processo no topo de uma lista decrescente construída com os valores de PM. $PM(i, j)$ está associado a um processo i e a um Conjunto j. O gerente desse conjunto deve selecionar o processador mais propício para receber o processo i. Antes de cada migração, sua viabilidade é verificada através do uso das seguintes informações: (i) carga externa nos processadores origem e destino; (ii) 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; (v) custos de migração. Dessa forma, MigBSP computa dois tempos: t_1 e t_2. t_1 representa a execução local do processo i, enquanto t_2 compreende sua execução em um outro processador e inclui os custos de migração. Para o candidato, um novo recurso é escolhido se $t_1 > t_2$.

3. AutoMig: Proposta para Auto-Selecionar os Candidatos a Migração

AutoMig é uma heurística para a seleção automática de processos para migração a cada chamada de reescalonamento. Ela pode eleger não somente um, mas uma coleção de processos. AutoMig auto-organiza os processos migráveis sem a intervenção do programador em ambos os níveis de aplicação e escalonador. Levando em consideração a função de decisão PM, AutoMig propõe uma solução para a sentença abaixo.

- **Sentença do Problema:** Dados n processos e uma lista do maior PM (Potencial de Migração) de cada um deles no momento do reescalonamento, o desafio consiste na criação e avaliação de no máximo n planos de escalonamento e escolher o mais apto entre aqueles que têm desempenho melhor que o mapeamento corrente.

AutoMig resolve a sentença acima usando os conceitos de *List Scheduling* e *Backtracking*. Primeiramente, é feito o ordenamento da lista de PM de maneira decrescente. Dessa forma, os testes começam pelo processo no topo visto que representa as melhores chances de ganhos com a migração. Num segundo momento, AutoMig processa n tentativas de escalonamento (onde n é o número de processos) através do incremento da movimentação de apenas um processo a cada novo plano. Essa ideia é baseada na técnica do *Backtracking*, onde cada candidato parcial é o pai de candidatos que diferem dele de

apenas um passo de escolha [Baritompa et al. 2007]. O Algoritmo 1 representa a abordagem de AutoMig em detalhes. A Figura 1 expande a explicação do algoritmo em um exemplo com 7 processos. Para cada processo i descrito em $PM(i, j)$, AutoMig emula sua execução aplicando uma predição pf com o intuito de medir o desempenho do novo escalonamento. Esse escalonamento é calculado através do teste de relocação do processo i para o Conjunto j denotado como parâmetro de PM.

Algoritmo 1 Escolhendo os processos com AutoMig no momento do reescalonamento

1: Cada processo computa PM (Potencial de Migração) localmente (veja Equação 1).
2: Cada processo passa o seu maior PM, junto com o número de instruções e um vetor que descreve suas ações de comunicação, para o seu Gerente de Conjunto.
3: Gerentes de Conjunto trocam os valores de PM de seus respectivos processos.
4: Gerentes de Conjunto criam uma lista decrescente baseada nos valores de PM. Tal lista possui n elementos, onde n é o número de processos.
5: Cada Gerente de Conjunto computa pf para os processos sob sua jurisdição. Tais dados serão úteis para mensurar o desempenho do mapeamento corrente.
6: **for** cada elemento partindo de 0 até $n - 1$ que estão na lista de $PM(i, j)$ **do**
7: O elemento considerado é analisado. Gerente do processo i envia dados para o Gerente do Conjunto j. Esse último é responsável por computar os dados para i.
8: O gerente no Conjunto destino escolhe um processador para processo i. A partir de agora, tal processo é considerado como migrado para o Conjunto j para fins de cálculo do algoritmo.
9: Cada Gerente de Conjunto calcula pf para os processos sob sua jurisdição.
10: Gerentes de Conjunto salvam dados de pf em um vetor.
11: **end for**
12: Gerentes de Conjunto trocam os seus dados e computam pf para o escalonamento corrente assim como para cada nível na lista de PM.
13: **if** $Min(pf)$ na lista de $PM < pf$ corrente **then**
14: Considerando a lista de PM, os processos pertencentes ao nível onde pf foi encontrado são selecionados para migração.
15: Gerentes notificam os processos eleitos que estão sob sua jurisdição para migrar
16: **else**
17: Migrações não acontecem.
18: **end if**

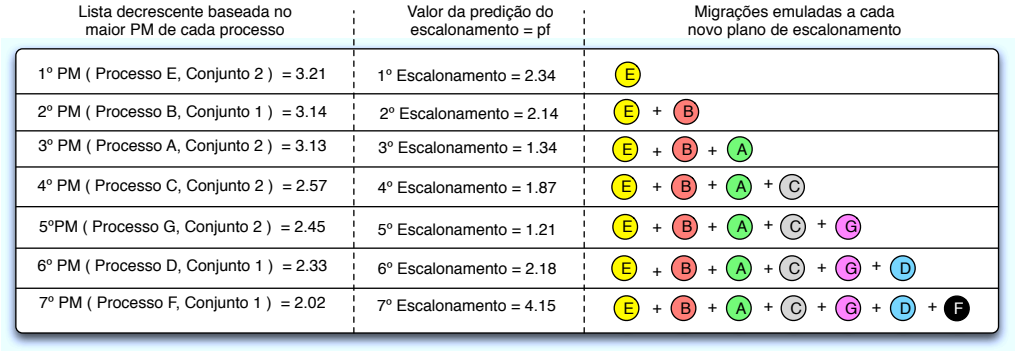

Figura 1. Exemplo da abordagem de AutoMig: Cada elemento na lista de PM produz um novo escalonamento através do incremento de uma migração

O algoritmo começa pela avaliação do escalonamento atual. Na sequência, AutoMig calcula pf para cada um dos níveis da lista de PM. Analisando o exemplo apresentado na Figura 1, o desempenho da predição do processo "A" no terceiro nível de PM leva em consideração a migração deste processo e o fato de que "E" e "B" também foram previamente migrados. A seleção final é obtida através da verificação do menor pf. Os processos pertencentes a essa predição são eleitos para migração caso esse novo mapeamento ultrapasse o desempenho do corrente. A principal parte de AutoMig é a sua função de predição pf. No momento do reescalonamento, cada processo passa as seguintes informações para seu gerente: (i) o maior valor de PM; (ii) um vetor com os custos de migração (métrica Mem) para cada Conjunto; (iii) um vetor o qual contém a quantidade de bytes em ações de comunicação para cada um dos Conjuntos. Esses dois últimos dados são capturados na última superetapa executada. pf considera ambas as partes de computação e comunicação com o intuito de emular o tempo despendido na superetapa. As duas são encontradas através das Equações 2 e 3, respectivamente. Em adição, pf trabalha com os custos de migração de processos para Conjuntos. Inicialmente, pf é usado para computar o desempenho do mapeamento corrente. Cada Gerente o computa para seus processos e salva os resultados localmente (passo 5 no Algoritmo 1).

O próximo passo, procede-se com o cálculo de pf para cada processo na lista de PM. Em cada nível, informações do processo alvo são transferidas para o Conjunto destino. A cargo de exemplo, dados do processo "E" são transferidos para o Conjunto 2 como ilustra a Figura 1. Assim, o gerente no Conjunto destino irá escolher o processador mais apto sob sua jurisdição para o processo e irá calcular ambas as Equações 2 e 3 para ele. Com o intuito de minimizar a comunicação *multicast* entre os gerentes a cada computação de pf, cada Gerente de Conjunto computa $Time_p$ e $Comm_p$ para os processos sob seu comando e salva os dados de um nível específico localmente. Depois de calcular o último pf, os gerentes trocam informações e calculam pf para cada nível da lista, bem como para o escalonamento corrente (passo 12 no Algoritmo 1).

A Equação 2 calcula $Time_p(i)$, onde i representa um processo específico. $Time_p(i)$ usa dados relativos ao poder de computação e dados de carga do processador na qual o processo i executa atualmente ou o processador que está sendo testado para reescalonamento. $cpu\_load(i)$ denota a média de carga da CPU nos últimos 15 minutos. Este intervalo de tempo foi adotado baseado nos estudos prévios de Vozmediano e Conde [Moreno-Vozmediano and Alonso-Conde 2005]. A Equação 3 mostra como é atingido o tempo máximo de computação ao considerar o processo i e o Conjunto j. Neste contexto, o Conjunto j pode ser o atual do processo i ou o Conjunto no qual o processo está sendo testado para migração. $T(k, j)$ se refere a taxa de transferência de 1 byte do Gerente do Conjunto j para outro gerente. Nessa mesma ideia, $Bytes(i, k)$ trabalha com o número de bytes transferidos pela rede entre o processo i e os processos que pertencem ao Conjunto k. Por fim, $Mig\_Costs(i, j)$ mostra os custos de migração relativos a transferência do processo i para o Conjunto j. A Equação 4 apresenta cada parte da função pf. É importante enfatizar que cada parte de pf pode considerar um processo i e um Conjunto j diferentes. A título de exemplo, um processo específico pode obter o maior tempo de computação, enquanto outro pode despender mais tempo nas ações de comunicação.

$$Time_p(i) = \frac{Instruction(i)}{(1 - cpu\_load(i)) \cdot cpu(i)} \quad (2)$$

$$Comm_p(i,j) = Max_k\ (\ \forall\ k \in Sets\ (Bytes(i,k).T(k,j))) \quad (3)$$

$$pf = Max_i\ (Time_p(i)) + Max_{i,j}\ (Comm_p(i,j)) + Max_{i,j}\ (Mig\_Costs(i,j)) \quad (4)$$

Depois de concluir o passo 12 do Algoritmo 1, cada Gerente de Conjunto conhece todos os valores de predição. Consequentemente, ele está apto a discernir sobre os processos que irão migrar. Uma ou mais migrações podem ocorrer caso pf para a situação corrente seja maior que aquela encontrada em quaisquer dos níveis da lista de PM. Se isso for o caso, cada Gerente de Conjunto verifica os processos migráveis sobre sua responsabilidade e os avisa sobre a tarefa de relocação. Os principais pontos positivos de AutoMig são: (i) a não necessidade de informar parâmetros adicionais em MigBSP sobre a seleção dos processos; (ii) minimizar comunicação *multicast* entre os gerentes com o intuito de reduzir os custos de escalonamento; (iii) possibilidade de selecionar uma coleção de processos para migração. Além disso, o desempenho final da aplicação irá depender do comportamento dos processos depois do reescalonamento. AutoMig assume que os processos irão reproduzir suas ações de computação e comunicação no computador destino. A questão que trata a regularidade dos processos já é tratada na computação de PM (ver [da Rosa Righi et al. 2010] para mais detalhes).

Kowk e Cheung [Kwok and Cheung 2004] classificam o tópico de balanceamento de carga em quatro classes: (i) política de localização; (ii) política de informação; (iii) política de transferência; (iv) política de seleção. AutoMig responde a última questão usando uma estratégia global [Zaki et al. 1997]. Para tanto, a lista do maior PM de todos os processos BSP é conhecida pelos Gerentes de Conjunto no momento da tentativa de migração. Dessa forma, a principal vantagem de um esquema global compreende uma melhor qualidade nas decisões de balanceamento de carga uma vez que todos os objetos estudados são considerados. Entretanto, a sincronização é a parte mais pesada nessa abordagem [Zaki et al. 1997]. Mas AutoMig tira proveito da organização do modelo BSP que já impõe por si só uma barreira entre os processos. Assim, não é necessário adicionar um custo a mais para implementar a ideia global de balanceamento de carga.

4. Metodologia de Avaliação

A avaliação passa pela simulação do funcionamento de uma aplicação BSP que realiza a compressão de imagens segundo a técnica de fractais. Tal técnica tem gerado interesse na comunidade de compressão de imagens como um possível competidor com formatos já estabelecidos como JPEG e Wavelets [Guo et al. 2009]. Um dos principais problemas na abordagem de fractal é a alta complexidade de compressão. Entretanto, a decodificação acontece num tempo muito menor [Xing 2008]. Tal técnica explora similaridades dentro das imagens, as quais são descritas por uma transformação contrativa da imagem. Essa operação consiste na transformação de blocos os quais aproximam partes menores da imagem por partes mais largas. As partes menores são chamadas de intervalo, enquanto as maiores de domínio. Todos os intervalos formam uma imagem. Os domínios são selecionados livremente dentro da imagem. A modelagem BSP considera a variação nos tamanhos do intervalo e do domínio, bem como no número de processos. Primeiramente, são computadas $\frac{t}{r}$ superetapas, onde $t \times t$ é o tamanho da imagem e r é o tamanho do lado do intervalo. O objetivo é computar um conjunto de intervalos em cada superetapa.

A cada superetapa, um intervalo é testado contra $8((\frac{t}{d})^2.\frac{1}{n})$ domínios, onde d representa o tamanho de um domínio e n o número de processos. Além disso, cada domínio envia $\frac{t}{r}$ intervalos antes de chamar a função de barreira. Tal quantidade deve ser multiplicada por 8 para encontrar o número de bytes transferidos pela rede.

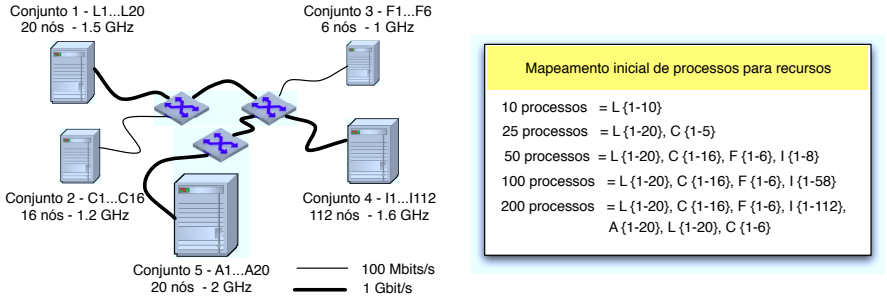

Figura 2. Topologia de testes com múltiplos clusters, descrição dos recursos de processamento e de rede, bem como mapeamento inicial processos-recursos

O principal objetivo da avaliação é observar o desempenho de MigBSP ao usar a heurística AutoMig. Baseado nesse princípio, foi aplicada simulação em três cenários: (i) execução da aplicação simplesmente; (ii) execução da aplicação com MigBSP sem aplicar migrações; (iii) execução da aplicação com MigBSP permitindo as migrações. O cenário ii compreende todos os cálculos de escalonamento mas não faz a transferência de processos. O cenário iii adiciona os custo de migração caso quaisquer processos sejam transferidos para novos recursos. A aplicação e o modelo foram desenvolvidos usando o simulador Simgrid (Módulo MSG) [Casanova et al. 2008]. Ele possui um caráter determinístico, onde os mesmos dados de entrada resultam sempre numa saída igual.

Para avaliar os cenários, foi montada uma infraestrutura com 5 Conjuntos os quais são descritos na Figura 2. Um Conjunto representa um *cluster* onde cada nó tem um único processador. A infraestrutura permite analisar o impacto da heterogeneidade nos algoritmos de AutoMig. Além disso, ela é uma adaptação dos recursos existentes atualmente na Universidade do Vale do Rio dos Sinos. Testes iniciais foram executados usando α igual a 4 e D igual a 0.5 e uma quantidade de processos que varia da seguinte maneira: 10, 25, 50, 100 e 200 processos. Seus mapeamentos iniciais para recursos podem ser vistos na Figura 2. Uma vez que a aplicação desempenha comunicações do processo i para o $i+1$, foi adotada a abordagem contínua na qual cada *cluster* é totalmente preenchido antes de completar o próximo [Pascual et al. 2009]. Além dos parâmetros previamente denotados, usou-se os valores de 40, 20 e 10 para o lado d do domínio e um intervalo obtido por $\frac{d}{2}$. A figura considerada é quadrada com lado 1000. Quanto menor o valor d, maior o número de domínios para serem testados em cada processo. Por fim, os custos de migração estão baseados em execuções com AMPI [Huang et al. 2006] em nossos *clusters*.

5. Analisando o Desempenho e as Decisões de AutoMig

A Tabela 1 apresenta os testes iniciais que ocorrem no tratamento dos valores de 40 e 20 para domínio e intervalo. Esta configuração habilita um número pequeno de domínios para serem computados em cada processo. Sendo assim, cada um requisita poucos recursos de processamento e suas migrações não são viáveis. Os valores de PM nessa situação são negativos devido ao baixo peso das ações de computação e comunicação se comparadas com os custos de migração. Consequentemente, ambos os tempos dos cenários ii e iii

são maiores que o do i. Nesse contexto, uma grande sobrecarga é imposta por MigBSP uma vez que a aplicação executa em aproximadamente 1 segundo.

Tabela 1. Resultados com 40 e 20 para domínio e intervalo (tempo em segundos)

Processos	Cenário i	Cenário ii	Cenário iii
10	1.20	2.17	2.17
25	0.66	1.96	1.96
50	0.57	2.06	2.06
100	0.93	2.44	2.44
200	1.74	3.41	3.41

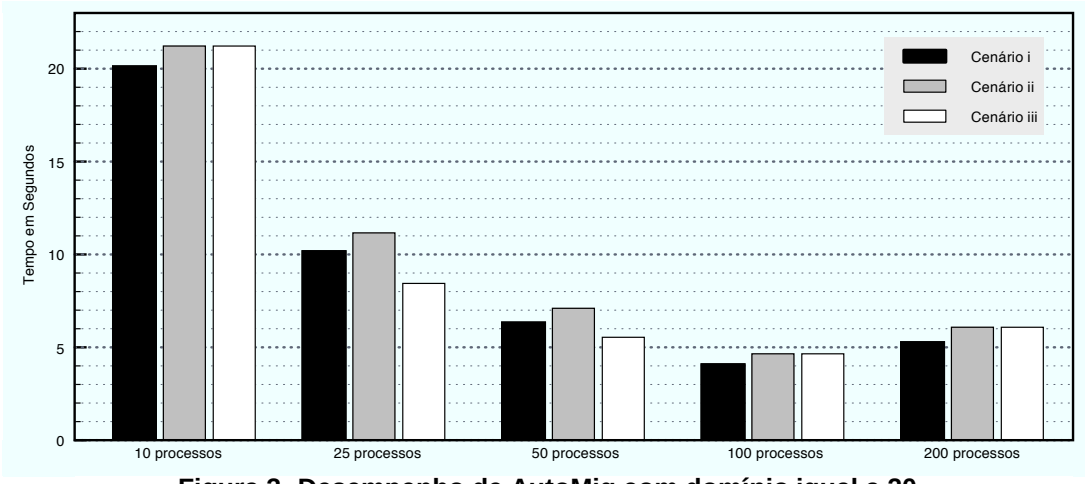

Figura 3. Desempenho de AutoMig com domínio igual a 20

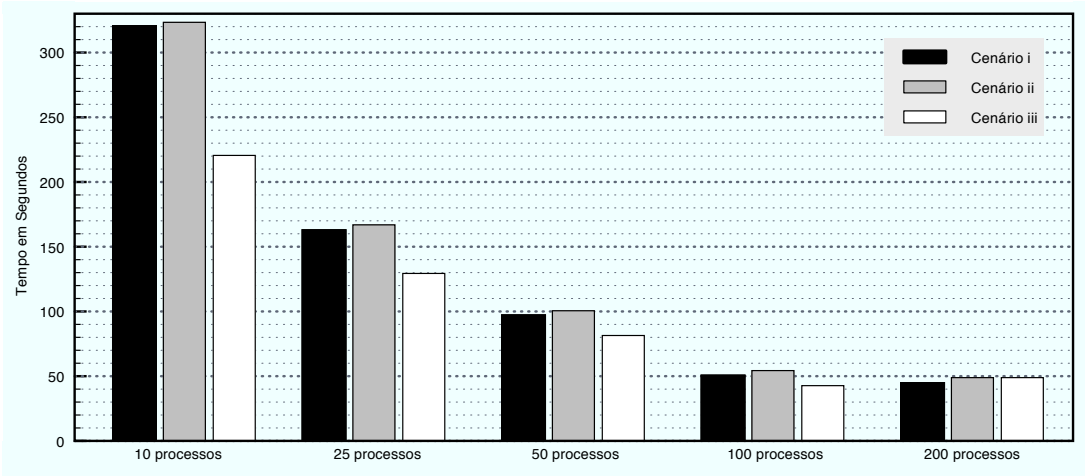

Figura 4. Resultados de AutoMig quando o trabalho para cada processo é aumentado. O gráfico ilustra experimentos com domínio 10 e intervalo 5

O número de domínios é aumentado ao lidar com o número 20 para o lado do domínio (veja gráfico da Figura 3). Como apresentado na execução prévia, migrações não acontecem com 10 processos. Eles estão balanceados e sua reorganização para um *cluster* mais rápido impõe um custo maior que os benefícios. Um valor de pf de 0.21 foi obtido para o mapeamento processos-recursos corrente ao usar 20 para domínio e 10 processos. Todas as predições na lista de PM são maiores de 0.21 e sua média alcança 0.38. Entretanto, essa configuração de domínio lança migrações com 25 e 50 processos. No primeiro caso, 5 processos do *cluster* C são movidos para o *cluster* mais rápido chamado

A. As decisões de AutoMig acarretam um ganho de 17.15% com o reescalonamento de processos nesse contexto. O último *cluster* mencionado recebe todos os processos do *cluster* F quando a aplicação executa com 50 processos. Essa situação mostra ganhos de 20.05% com as migrações. Embora 14 nós do *cluster* mais rápido A estejam livres, AutoMig não seleciona processos para executar neles pois o modelo BSP apresenta uma barreira de sincronização. Por exemplo, ainda que 14 migrações ocorram do *cluster* C para o A, um grupo de processo permanecerá no *cluster* mais lento e irão determinar o tempo da superetapa. Por fim, uma vez que o grão de trabalho diminui na medida que aumenta-se os processos, execuções com 100 e 200 deles não mostraram migrações.

A aplicação BSP demonstrou um bom nível de desempenho com domínio igual a 10, como pode ser visto na Figura 4. O grão de computação aumenta exponencialmente com esta configuração. Isso pode ser observado principalmente na execução de 10 processos, onde todos eles migram para o *cluster* A. Considerando que $8((\frac{t}{d})^2.\frac{1}{10})$ expressa o número de domínios mapeados para cada um dos 10 processos, essa expressão é igual a 500, 2000 e 8000 ao lidar com os valores para domínios iguais a 40, 20 e 10. Usando o valor 10 para domínio e processos, o escalonamento corrente produziu um pf igual a 1.62. Os valores de pf para a lista de PM são vistos a seguir:

- $pf[1..10] = \{1.79, 1.75, 1.78, 1.79, 1.81, 1.76, 1.74, 1.82, 1.78, 1.47\}$.

Considerando o primeiro até o nono pf, pode-se verificar que embora alguns processos rodem mais rápido num *cluster* mais apropriado, existem outros que permanecem no *cluster* com processamento mais lento. Esse último grupo não permite ganhos de desempenho devido a própria modelagem BSP. Essa situação muda no momento de testar o décimo pf. Esse teste leva em questão a migração dos 10 processos para o *cluster* mais rápido e gera um ganho de 31.13% ao comparar os cenários i e iii (ver Figura 5).

Os processos do *cluster* C são movidos para o A nos testes que envolvem 25 processos e um domínio de 10. Nesse caso, os 20 outros processos ficam no *cluster* L porque não existem nós livres suficientes no *cluster* mais rápido. Uma possibilidade é explorar dois processos em um nó do *cluster* A (cada nó possui 2 GHz) mas AutoMig não considera essa ideia pois cada nó do *cluster* L possui 1.2 GHz. Considerando o aumento no número de domínios, as migrações com 100 processos se tornam viáveis e representam uma melhora de 14.95% no desempenho. Por outro lado, o mapeamento inicial de 200 processos ainda permanece na mesma posição e acarreta uma sobrecarga de 7.64% na execução da aplicação. Tal percentagem é obtida ao comparar os cenários i e ii.

Os testes anteriores deixam claro que quanto maior o peso de computação por processo, melhores serão os ganhos com o reescalonamento de processos. Nesse caminho, testou-se AutoMig com domínio e intervalo menores como expressado na Tabela 2. Esta tabela mostra o comportamento para 10 e 25 processos. Ganhos de cerca de 31.6% e 19.81% foram obtidos ao aplicar AutoMig. Em adição, foi verificada uma sobrecarga menor que 1%. Analisando os resultados, quanto maior o tempo da aplicação, menor o impacto de AutoMig sobre ela. Ainda, verificou-se que os benefícios com as migrações se tornam praticamente constantes se forem comparadas com as execuções com domínio 10 e 4. Ao dobrar o número de processos, o tempo da aplicação não cai pela metade. Existe um limite onde a inclusão de novos processos não gera ganhos no tempo de execução. Considerando a questão da escalabilidade, MigBSP (com AutoMig) obteve comportamentos similares se comparado com aqueles capturados do cenário i (ver Figuras 3 e 5).

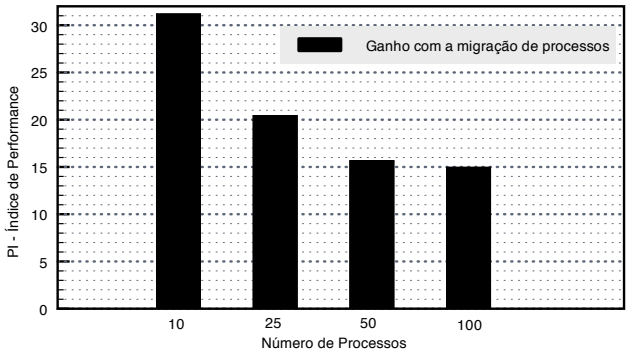

Figura 5. Ganhos com domínio 10. Índice PI = $(\frac{cenario\ i - cenario\ iii}{cenario\ i} * 100)$

Tabela 2. Resultados com 4 e 2 para domínio e intervalo (tempo em segundos)

Processos	Cenário i	Heurística Velha		AutoMig	
		Cenário ii	Cenário iii	Cenário ii	Cenário iii
10	12500.51	12511.87	9191.72	12523.22	8555.29
25	6250.49	6257.18	5311.54	6265.38	5011.77

A Tabela 2 também mostra uma análise comparativa das duas heurísticas de seleção de MigBSP. A título de simplificação, foi nomeada a heurística que seleciona um processo para migração como "Heurística Velha". Embora ambos tenham obtido bons índices de desempenho, AutoMig se sobressai em relação a outra em aproximadamente 8%. Por exemplo, 5 processos são migrados já na primeira tentativa de migração quando 25 processos são testados. Neste caso, todos os processos que estavam executando no *cluster* C são passados para o *cluster* A. Esta reorganização sugerida por AutoMig no começo da aplicação fornece um menor tempo para conclusão da aplicação. Por outro lado, 5 chamadas de reescalonamento são necessárias para atingir a mesma configuração com a Heurística Velha. Por fim, a heurística AutoMig impõe maiores sobrecargas se comparado com a sua antecessora (próximo a 1%). Esta situação foi esperada pois duas comunicações *multicast* entre os Gerentes são realizadas por AutoMig em seu algoritmo.

6. Trabalhos Relacionados

Vadhiyar e Dongarra [Vadhiyar and Dongarra 2005] apresentam suporte a migrações e adaptatividade no sistema GrADS. O custo de migração é considerado como um valor fixo e os ganhos com o reescalonamento são baseados no tempo restante de execução. Assim, GrADS trabalha com aplicações cujas partes sejam conhecidas previamente. Em adição, o mesmo problema é visto em dois trabalhos relacionados. Sanjay e Vadhiyar [Sanjay and Vadhiyar 2009] apresentam um algoritmo de escalonamento que considera um gráfico 3D unindo informações de CPU e largura de banda para a seleção dos recursos com maior carga atual. O segundo trabalho compreende os esforços de Ding [Ding et al. 2009]. Ele criou uma heurística para partição de tarefas que minimiza o tempo de conclusão da carga de trabalho. Sua abordagem conhece o tamanho do problema antes da execução do programa. Alternativamente, AutoMig usa somente dados coletados em tempo de execução para suas tomadas de decisões.

Chen et al. [Chen et al. 2008] propuseram remapeamento de processos com redução do custo. A heurística desenvolvida permite somente movimentos de processos entre a máquina com maior carga e uma outra. Além disso, diferente de AutoMig, a

solução desses autores não considera as ações de comunicação para a seleção. Silva et al. [da Silva e Silva et al. 2010] expuseram o gerenciamento de recursos sobre o *middleware* InteGrade. Eles apresentaram um *grid* como uma coleção de *clusters*, onde cada qual possui o seu próprio gerente (CM - *Cluster Manager*). Análogo a proposta de AutoMig, cada CM é responsável por capturar dados de um *cluster* e trocá-los com os demais CMs.

Quanto a migração em aplicações BSP, destacam-se as bibliotecas PUBWCL e PUB. PUBWCL tira proveito de ciclos ociosos de computadores na Internet [Bonorden et al. 2005]. Seus algoritmos usam dados sobre os recursos de computação, bem como o tempo de processamento dos processos. A PUB [Bonorden 2007], por sua vez, propõe estratégias de balanceamento centralizada e distribuída. Na centralizada, todos os nós enviam dados sobre seu poder de CPU para um mestre. Esse verifica as informações e seleciona um processo para transferência. Na abordagem distribuída, cada nó escolhe c outros nós aleatóriamente e os questiona sobre a sua carga. Um processo é migrado se a carga mínima dos c nós analisados for menor que a carga do nó que realizou o teste. Os pontos negativos de ambas abordagens são: (i) uso isolado de dados de computação; (ii) um processo é migrado por ativação do rebalanceamento.

7. Conclusão

Considerando que o estilo de computação em fases de BSP é uma organização frequente na escrita de programas paralelos de sucesso [Bonorden 2007, De Grande and Boukerche 2011, Hendrickson 2009], AutoMig emerge como uma alternativa para selecionar seus processos para executar em recursos mais apropriados sem a interferência de desenvolvedores. A principal contribuição de AutoMig aparece na sua função de predição pf. Ela é aplicada para o escalonamento corrente bem como para cada nível da lista baseada no PM dos processos. Cada elemento dessa lista informa um novo escalonamento através do incremento de uma reposição de processo. pf considera a carga nos Conjuntos e na rede, estima os processos mais lentos considerando suas atividades de computação e comunicação e adiciona o tempo de transferência do processo testado. O esquema de balanceamento de carga de AutoMig usa uma abordagem global, onde são usados dados de todos os processos. Ao invés de pagar um custo adicional de sincronização para adquirir informações de escalonamento, AutoMig tira proveito do conceito de superetapa do próprio modelo BSP, onde sempre há uma barreira após ações de comunicação entre os processos.

AutoMig e uma aplicação foram desenvolvidos usando o simulador Simgrid. A aplicação trabalha com um parâmetro chamado domínio que torna possível a criação de situações de carga mais facilmente. Uma vez que a aplicação é CPU-Bound, quanto menor o tamanho do domínio, maior o tempo de execução da aplicação e a probabilidade de migração. Os resultados provaram isso, indicando ganhos de até 17.15% e 31.13% para domínios igual a 20 e 10. Particularmente, os resultados revelaram um dos principais pontos positivos de AutoMig na seleção dos processos a migração. Ele pode eleger todos os processos de um *cluster* devagar para executar rapidamente em outro mais apropriado. Entretanto, muitas vezes o *cluster* mais rápido possui poucos nós livres do que o número de candidatos a migração. Assim, AutoMig demonstrou que as migrações são inviáveis nessa situação, devido as regras de execução de uma superetapa BSP. Uma superetapa BSP possui uma barreira que sempre espera pelo processo mais lento (neste caso, os processos que irão permanecer no *cluster* mais lento).

Por fim, trabalhos futuros compreendem o uso de AutoMig em um *middleware* para a computação em nuvem. Este *middleware* irá trabalhar no auto-dimensionamento de recursos para execução de aplicações paralelas. Considerando que cada aplicação tem seu próprio SLA (*Service Level Agreement*), AutoMig surge como a primeira iniciativa para reescalonamento em tempo de execução quando o SLA falha. Caso as migrações não resolverem a questão do desempenho, mais recursos são alocados num segundo momento.

Referências

Baritompa, W., Bulger, D. W., and Wood, G. R. (2007). Generating functions and the performance of backtracking adaptive search. *J. of Global Optimization*, 37:159–175.

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 *Int. Parallel and Distrib. Processing Symp.*, 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.

Chen, L., Wang, C.-L., and Lau, F. (2008). Process reassignment with reduced migration cost in grid load rebalancing. *Parallel and Distrib. Processing*, 2008, pages 1–13,IEEE.

da Rosa Righi, R., Pilla, L. L., Carissimi, A., Navaux, P. A., and Heiss, H.-U. (2010). Observing the impact of multiple metrics and runtime adaptations on bsp process rescheduling. *Parallel Processing Letters*, 20(2):123–144.

da Silva e Silva, F. J., Kon, F., Goldman, A., Finger, M., de Camargo, R. Y., Filho, F. C., and Costa, F. M. (2010). Application execution management on the integrade opportunistic grid middleware. *J. Parallel Distrib. Comput.*, 70(5):573–583.

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

Ding, D., Luo, S., and Gao, Z. (2009). A dual heuristic scheduling strategy based on task partition in grid environments. In *Int. Joint Conference on Computational Sciences and Optimization*, pages 63–67, IEEE.

Duselis, J., Cauich, E., Wang, R., and Scherson, I. (2009). Resource selection and allocation for dynamic adaptive computing in heterogeneous clusters. In *Cluster Computing and Workshops, 2009. CLUSTER '09. IEEE International Conference on*, pages 1 –9.

El Kabbany, G., Wanas, N., Hegazi, N., and Shaheen, S. (2011). A dynamic load balancing framework for real-time applications in message passing systems. *International Journal of Parallel Programming*, 39:143–182. 10.1007/s10766-010-0134-5.

Guo, Y., Chen, X., Deng, M., Wang, Z., Lv, W., Xu, C., and Wang, T. (2009). The fractal compression coding in mobile video monitoring system. In *WRI International Conference on Comm. and Mobile Computing*, pages 492–495, IEEE.

Hendrickson, B. (2009). Computational science: Emerging opportunities and challenges. *Journal of Physics: Conference Series*, 180(1):012013.

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, ACM Press.

Kwok, Y.-K. and Cheung, L.-S. (2004). A new fuzzy-decision based load balancing system for distributed object computing. *J. Parallel Distrib. Comput.*, 64(2):238–253.

Moreno-Vozmediano, R. and Alonso-Conde, A. B. (2005). Influence of grid economic factors on scheduling and migration. In *High Perf. Comp. for Computational Science*, volume 3402 of *Lecture Notes in Comp. Science*, pages 274–287. Springer.

Nascimento, A. P., Sena, A. C., Boeres, C., and Rebello, V. E. F. (2007). Distributed and dynamic self-scheduling of parallel mpi grid applications: Research articles. *Concurr. Comput. : Pract. Exper.*, 19(14):1955–1974.

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

Qin, X., Jiang, H., Manzanares, A., Ruan, X., and Yin, S. (2010). Communication-aware load balancing for parallel applic. on clusters. *IEEE Trans. Comput.*, 59(1):42–52.

Sanjay, H. A. and Vadhiyar, S. S. (2009). A strategy for scheduling tightly coupled parallel applications on clusters. *Concurr. Comput. : Pract. Exper.*, 21(18):2491–2517.

Silva, R. E., Pezzi, G., Maillard, N., and Diverio, T. (2005). Automatic data-flow graph generation of mpi programs. In *SBAC-PAD '05: Proceedings of the 17th International Symposium on Computer Architecture on High Performance Computing*, pages 93–100, Washington, DC, USA. IEEE Computer Society.

Tanenbaum, A. (2003). *Computer Networks*. Prentice Hall, New Jersey, 4th ed.

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

Xhafa, F. and Abraham, A. (2010). Computational models and heuristic methods for grid scheduling problems. *Future Gener. Comput. Syst.*, 26(4):608–621.

Xing, C. (2008). An adaptive domain pool scheme for fractal image compression. *Education Technology and Training and Geoscience and Remote Sensing*, 2:719–722.

Yang, C.-T. and Chou, K.-Y. (2009). An adaptive job allocation strategy for heterogeneous multiple clusters. In *Int. Conf. on Computer and Information Technology - Volume 02*, pages 209–214, IEEE.

Zaki, M. J., Li, W., and Parthasarathy, S. (1997). Customized dynamic load balancing for a network of workstations. *J. Parallel Distrib. Comput.*, 43(2):156–162.