Avaliação de Estratégias de Balanceamento de Carga do Tipo Mestre-Escravo para Aplicações SPMD em *Clusters* e *Grids* Computacionais

Andre L. de Oliveira
UFRJ – COPPE Sistemas
Rio de Janeiro, RJ, Brasil
aoliveira100@gmail.com

Alexandre Plastino[1]
Simone de L. Martins[2]
UFF – Inst. de Computação
Niterói, RJ, Brasil
{plastino[1],simone[2]}@ic.uff.br

Inês Dutra
Universidade do Porto
Depto. de Ciência de Comp.
Porto, Portugal
ines@dcc.fc.up.pt

Resumo

O desempenho de programas paralelos é fortemente influenciado por diferentes fatores dinâmicos de desequilíbrio de carga. A utilização de um algoritmo de balanceamento de carga adequado é essencial para a redução dos efeitos causados por esses fatores de desequilíbrio. Neste trabalho, avalia-se o desempenho de diversas estratégias de balanceamento quando executadas em aplicações SPMD com diferentes características. Os experimentos foram realizados em um cluster e em um grid computacional e foram considerados diversos fatores que podem ocasionar desequilíbrios em um ambiente computacional, tais como: capacidade de processamento, presença de carga externa à aplicação e velocidade de comunicação.

1. Introdução

O modelo de programação paralela SPMD (*Single Program Multiple Data*) tem sido amplamente utilizado devido à facilidade de desenvolvimento e controle de um código único (*Single Program*) e ao fato deste modelo ser capaz de representar uma grande quantidade de problemas. Nesse modelo, o mesmo programa é executado, sem sincronismo no nível de instrução, nos diferentes processadores, manipulando conjuntos distintos de dados – que caracterizam as tarefas.

O desempenho de uma aplicação SPMD pode ser afetado por diversos fatores que causam o desequilíbrio de carga, tais como: a heterogeneidade da arquitetura paralela, o desconhecimento da quantidade de processamento em cada tarefa, a criação dinâmica de tarefas, além da heterogeneidade e da variação da carga externa à aplicação em um ambiente não dedicado. A utilização de uma estratégia adequada de balanceamento de carga é fundamental para a redução do efeito desses fatores [13].

O objetivo deste trabalho é apresentar os principais resultados dos experimentos realizados em [3] com diversas estratégias de balanceamento de carga. Dez estratégias distintas foram avaliadas, para diferentes aplicações SPMD, em ambientes computacionais com diferentes características. Três aplicações foram utilizadas: uma aplicação que calcula a dispersão térmica em meios porosos, uma aplicação para renderização de imagens que utiliza *ray tracing* e uma aplicação sintética para simular aplicações com diferentes características. Em relação às características da aplicação, consideraram-se fatores como o tempo de computação e o tamanho de cada tarefa, além da quantidade total e da heterogeneidade das tarefas. Os experimentos foram realizados em um *cluster* e, posteriormente, em um *grid* computacional constituído por dois *clusters* geograficamente distantes. A análise de desempenho das estratégias nesses ambientes considerou fatores como: quantidade de máquinas, capacidade de processamento, presença de carga externa à aplicação e tempo de comunicação.

A maioria das estratégias avaliadas adota a abordagem mestre-escravo, amplamente utilizada para resolver problemas de balanceamento de carga [4, 8, 12, 13]. O problema de definição do tamanho dos blocos de tarefas enviados pelo mestre aos seus escravos também foi objeto de estudo. Nesse contexto, foram propostas e implementadas variações de estratégias existentes na literatura que utilizam uma abordagem de redução de bloco [9, 10, 15].

Este trabalho está organizado da seguinte forma. Na Seção 2, são apresentadas as ferramentas de apoio à implementação, execução e avaliação das estratégias de balanceamento de carga. Na Seção 3, as estratégias de balanceamento são descritas. Na Seção 4, são apresentadas as aplicações SPMD implementadas. Na Seção 5, são analisados os resultados dos experimentos computacionais. Finalmente, na Seção 6, são apresentadas as conclusões.

2. Ferramentas Utilizadas

SAMBA e SAMBA-Grid: A ferramenta SAMBA-Grid é um *framework* para desenvolvimento de aplicações paralelas SPMD com balanceamento de carga em *grids* computacionais. Foi desenvolvida a partir da ferramenta SAMBA criada para ser utilizada em *clusters* [13]. Para se gerar uma aplicação SPMD, o usuário tem que desenvolver, basicamente, o código comum que será executado nos diversos processadores sobre os diferentes conjuntos de dados. A ferramenta disponibiliza uma biblioteca de algoritmos de balanceamento de carga, que possibilita a geração automática de diferentes versões de um mesmo programa paralelo, bastando recompilar o programa com a opção de balanceamento desejada.

Network Weather Service(NWS): Esta ferramenta permite realizar o monitoramento periódico dos principais recursos de um sistema computacional distribuído. Processos sensores são executados nas máquinas onde se deseja monitorar recursos como: CPU, rede e memória.

Playload: Esta ferramenta [6] objetiva reproduzir a variação de carga de trabalho em processadores visando emular ambientes dinâmicos de concorrência. Para isso, monitora a média de carga de um processador registrando em um arquivo de *traces* que serve como entrada para o *Playload* quando se desejar reproduzir esta mesma variação de carga em um outro processador. Cada arquivo de *traces* utilizado neste trabalho [11] tem uma característica particular e é classificado segundo dois parâmetros. O primeiro se refere ao consumo médio de CPU e o segundo à variação deste consumo no decorrer do tempo (L - Low ou H - High).

Traffic Control (TC): Esta ferramenta permite configurar o compartilhamento das filas de envio de pacotes das interfaces de rede. Neste trabalho, foram utilizadas as funções do TC responsáveis por embutir retardos nos envios de pacotes para emular canais de comunicação com latências de rede mais elevadas.

3. Estratégias de Balanceamento de Carga

As estratégias de balanceamento de carga avaliadas neste trabalho são: duas estáticas, uma Mestre-Escravo com Bloco Fixo e estratégias do tipo Mestre-Escravo com Redução de Bloco. A idéia principal dessas últimas é distribuir blocos de tamanho maior no início da execução e, à medida que o mestre recebe novas solicitações de tarefas, o tamanho dos blocos vai reduzindo gradativamente até o final da execução da aplicação. O objetivo principal é diminuir o *overhead* de comunicação, sem gerar desbalanceamentos e tirar do usuário a responsabilidadede por definir o tamanho do bloco.

Estático Simples: Trata-se da estratégia mais trivial, pois é caracterizada apenas por uma política de distribuição que divide igualmente o conjunto inicial de tarefas entre todos os processadores no início da execução.

Estático com NWS: Consiste em uma variação do Estático Simples que leva em consideração o peso W_p associado a cada processador p, que é calculado utilizando a velocidade de seu processamento e a fração de CPU disponível obtida pelo NWS. O tamanho do lote L_p de tarefas destinado a um processador p é calculado pela fórmula $L_p = W_p \times N$, onde N é o número inicial de tarefas.

Mestre-Escravo com Bloco Fixo: Esta é uma estratégia de balanceamento amplamente utilizada [4, 8, 13]. Neste algoritmo, o processador mestre, que possui o conjunto inicial de tarefas, distribui para os seus escravos blocos de tamanhos fixos de tarefas. Toda vez em que um escravo termina de executar um bloco recebido, solicita um novo bloco ao mestre. Este processo se repete até que não existam mais tarefas pendentes de execução no mestre.

MERB1 – Guided Self-Scheduling: Proposta em [15], consiste em uma estratégia do tipo mestre-escravo com redução de bloco, na qual o tamanho do i-ésimo bloco B_i de tarefas destinado a um escravo é definido por $B_i = \left\lceil \left(1 - \frac{1}{P}\right)^i \times \frac{N}{P} \right\rceil$, onde i indica o número da solicitação, P o número de processadores escravos e N o número inicial de tarefas.

MERB2 – Factoring: Proposta em [9], consiste em distribuir as tarefas a partir de lotes formados por P blocos de mesmo tamanho, onde P é o número de processadores escravos. O tamanho de um bloco B_p dentro do i-ésimo lote é definido por $B_p = \left\lceil \left(\frac{1}{2}\right)^{i+1} \times \frac{N}{P} \right\rceil$, onde i indica o número de identificação do lote vigente, p o processador escravo que irá receber o bloco e N o número de tarefas. A cada P requisições por tarefas o identificador i do lote será incrementado e os blocos deste lote reduzirão de tamanho em relação aos blocos do lote anterior. Todos os blocos de um mesmo lote terão sempre o mesmo tamanho.

MERB3 – Weighted Factoring Simples: Proposta inicialmente em [10] e aperfeiçoada neste trabalho, é uma variação da estratégia *Factoring* para um sistema de multicomputadores. O Algoritmo 1, apresentado de forma simplificada, implementa esta estratégia e será executado todas as vezes que um escravo solicitar tarefas ao mestre. O peso (W_p) é dado pela velocidade de processamento de um escravo p, obtida a partir do arquivo de sistema do Unix (*/proc/cpuinfo*). Neste algoritmo, P é o número de escravos, N é o número de tarefas e $PercTar_p$ é o percentual de tarefas destinado a um escravo p, o qual é calculado apenas no início da execução, a partir do peso W_p. No inicio da execução $IDlote = 1$ e $tamanhoDoLote = 0$.

MERB4 – Weighted Factoring com NWS Estático: Esta estratégia é uma variação da MERB3 que utiliza os valores de disponibilidade de CPU, fornecidos pelo NWS, como parâmetros para definir pesos (W_p) mais eficientes.

```
1 início
2     se tamanhoDoLote <= 0 então
3         tamanhoDoLote ⟵ ⌈(1/2)^IDlote × N⌉;
4         tamanhoInicialDoLote ⟵ tamanhoDoLote;
5         IDlote ⟵ IDlote + 1;
6     bloco ⟵ tamanhoInicialDoLote × PercTar_p;
7     tamanhoDoLote ⟵ tamanhoDoLote − bloco;
8     retorna bloco;
9 fim
```

Algoritmo 1: Algoritmo MERB3

O peso de um processador é calculado pela fórmula $W_p = clockCpu_p \times availableCpu_p$, onde $clockCpu_p$ é o valor do *clock* do escravo p obtido a partir do arquivo de sistema do Unix e $availableCpu_p$ é a medição NWS que fornece a fração de *clocks* disponíveis para um novo processo ser executado no escravo p.

MERB5 – Weighted Factoring com NWS Dinâmico: Consiste em uma variação da MERB4 que atualiza os pesos dos processadores escravos durante a execução da aplicação, de forma a considerar variações de carga no ambiente. O cálculo do peso W_p é o mesmo da estratégia MERB4 e é feito no início da execução da aplicação e todas as vezes que um novo lote vigente é recalculado. Porém, apenas o cálculo de peso feito no início da execução utiliza a medição NWS $availableCpu_p$. As atualizações de peso utilizam a medição $currentCpu_p$, que fornece a disponibilidade de CPU para um processo que já esteja em execução.

MERB6 – Weighted Factoring with Dynamic Feedback: Proposta neste trabalho, consiste em uma variação da MERB5, onde o peso W_p é determinado pelo tempo médio de computação de uma tarefa (TMCT), retornado por um escravo durante a execução da aplicação. No início da execução, quando ainda não se conhece o TMCT dos escravos, adota-se uma política de distribuição fixa de tarefas. Todas as vezes que um escravo solicita novas tarefas ao mestre, envia junto o TMCT referente ao bloco executado imediatamente antes da solicitação. O mestre registra este valor e verifica se todos os demais escravos também já retornaram seus TMCT ao menos uma vez. Em caso afirmativo, o algoritmo adotará, daí em diante, uma abordagem com redução de bloco de forma semelhante a usada pela estratégia MERB5, onde o peso da cada escravo será o TMCT mais atual. Em caso negativo, seguirá a distribuição sem considerar a capacidade de processamento dos escravos. A implementação deste algoritmo seguiu duas abordagens distintas: a primeira considerando o TMCT como sendo apenas o tempo médio de execução de uma tarefa e a segunda também levando em conta a latência de rede para enviar uma tarefa do mestre para um escravo que é obtida a partir da medição *latencyTcp* do NWS.

Mestre-Escravo Hierárquico (MEH) Esta estratégia discutida em [1, 7] e implementada em [2] consiste em uma variação da estratégia mestre-escravo com bloco fixo que utiliza uma topologia hierárquica de processadores. Nesta política, a raiz da hierarquia representa o mestre global, as folhas da hierarquia representam os escravos e os nós internos representam submestres. O mestre global distribui as tarefas sob demanda entre os seus filhos. Cada submestre também distribui sob demanda suas tarefas aos seus filhos e, sempre que estas são concluídas, este solicita mais tarefas ao seu mestre. Cada escravo, após executar as tarefas recebidas, solicita mais tarefas ao seu submestre.

4. Aplicações SPMD

Termions: A aplicação dos *Termions* [16] implementa a resolução de um problema da área de física que tem por objetivo calcular a dispersão térmica em meios porosos. A dispersão térmica é avaliada pelo movimento de partículas hipotéticas, denominadas termions. Cada termion possui uma quantidade fixa de energia e é representado por uma posição no espaço. Esta posição determina em que tipo de meio a partícula se encontra: sólido ou fluido. Inicialmente, um número grande de termions é liberado e, a cada iteração, a posição de todos os termions é atualizada de acordo com o tamanho do seu passo e direção. A definição da direção do movimento é escolhida aleatoriamente e o tamanho de cada passo depende das propriedades do meio no qual a partícula está e da velocidade de fluxo do fluido, caso ela se encontre na parte fluida. Após a realização de um certo número de passos, as partículas atingem suas posições finais e se obtém a dispersão térmica final. Cada termion percorre diferentes caminhos e a avaliação do caminho total realizado por um termion constitui uma tarefa do programa paralelo.

Ray Tracing Distribuído: A aplicação *Ray Tracing* distribuído [5] consiste em um método de renderização de imagens que se baseia no princípio natural de propagação da luz. A abordagem distribuída trata-se de um aperfeiçoamento do algoritmo de *Ray Tracing* tradicional [17] que subdivide cada *pixel* da imagem em *subpixels* onde, para *cada subpixel*, é executado o algoritmo tradicional. Uma subdivisão de *pixels* na ordem 2×2, por exemplo, produz quatro *subpixels*. Como o cálculo de cor em cada *subpixel* é feito de forma independente, esta aplicação pôde ser facilmente paralelizada como SPMD. Uma imagem a ser renderizada pode ser constituída por diversos objetos que compõem a cena. As tarefas desta aplicação tendem a ser heterogêneas, pois o tempo de computação de cada *pixel* dependerá da sua localização na imagem. Os *pixels* localizados em região de fundo basicamente não consomem muito tempo de computação, enquanto que, os *pixels* localizados em regiões de objetos demandam mais processamento.

Sintética: A aplicação Sintética foi implementada com o objetivo de permitir uma análise do desempenho das estratégias de balanceamento de carga quando executadas em aplicações com um maior número de tarefas e com tarefas

de tamanho maior. Cada tarefa nesta aplicação consiste na execução de um *loop*, onde em cada iteração é executada uma operação aritmética simples. Os seguintes parâmetros podem ser configurados: número total de tarefas, tempo de computação de cada tarefa e a quantidade de bytes associados a uma tarefa.

5. Experimentos Computacionais

Nesta seção, serão apresentados os experimentos realizados no *cluster* e no *grid*. Para analisar o desempenho das estratégias, foi calculado o índice de desbalanceamento de carga (IDC) [12, 14] definido por $IDC = \frac{\sum_{i=1}^{i=P}(t_f - t_i)}{(P-1)} / t_f$, onde P é a quantidade total de processadores, t_i é o tempo final de execução do i-ésimo processador e t_f é o tempo de execução do último processador a terminar suas tarefas.

5.1. Experimentos Realizados no *Cluster*

Os experimentos no *cluster* utilizaram até 16 processadores com a mesma configuração: Pentium 1715 MHz, interconectados por uma rede local *Fast Ethernet*, com a versão 7.0.4 do *mpilam* instalada. As estratégias foram executadas utilizando-se duas aplicações: *Termions* e *Ray Tracing* implementadas utilizando-se a ferramenta *SAMBA*. Para ambas variou-se o número de processadores (4, 8 e 16) para analisar a escalabilidade. Todas as estratégias foram avaliadas e estão identificadas como: Est(Estático Simples), EstNWS(Estático com NWS), MEBFX (MEBF com bloco de X tarefas) e MERB1 a MERB6. O comportamento das estratégias foi avaliado em cinco ambientes, descritos na Tabela 1. Cada ambiente é identificado por oito letras, onde cada par de letras indica o tipo de *trace* executado em uma, duas ou quatro máquinas, dependendo da quantidade total de máquinas utilizadas nos experimentos (4, 8 ou 16).

A aplicação dos *Termions* foi executada com 1000 tarefas e foram configuradas propriedades físicas do meio de forma que as tarefas levassem aproximadamente o mesmo tempo de execução. A aplicação *Ray Tracing* foi executada para duas imagens distintas, de tamanho 500×500 *pixels*, denominadas *5balls* e *room*. Cada *pixel* das imagens foi dividido em 16 sub-regiões (*subpixels*).

Os experimentos no ambiente SC (Figuras 1 e 5) evidenciam a característica homogênea das tarefas dos *Termions* e da renderização da imagem *5balls*, pois a maioria das estratégias apresentou um comportamento semelhante. Pode-se observar a escalabilidade nas duas aplicações, pois os tempos foram reduzidos à metade nas execuções de 4 para 8 processadores e de 8 para 16. As execuções das estratégias Est e EstNWS com a aplicação dos *Termions* obtiveram desempenhos semelhantes, indicando que o EstNWS distribuiu as tarefas da mesma forma que o Est, após verificar que os processadores estavam completamente disponíveis. Essas estratégias obtiveram um desempenho superior ao

ID do Ambiente	Descrição	Nº de Execuções
SC	Todos os processadores estão exclusivos e sem a presença de carga externa à aplicação.	*Termions* (3 execuções) / *Ray Tracing* (5 execuções)
CC-0.1.2.3	A cada 4 processadores tem-se: um processador com 0 carga, um com 1 carga, um com 2 cargas e um com 3.	*Termions* (3 execuções) / *Ray Tracing* (5 execuções)
CC-LL.LH.HL.HH	Utiliza o playload para gerar as cargas externas e a cada 4 processadores tem-se: um processador com carga de *traces* do tipo LL, um com LH, um com HL e um com HH.	*Termions* (20 execuções) / *Ray Tracing* (20 execuções)
CC-LL.LL.HL.HL	Utiliza o playload para gerar as cargas externas e a cada 4 processadores tem-se: dois processadores com cargas de *traces* do tipo LL e os outros dois com HL.	*Termions* (20 execuções) / *Ray Tracing* (20 execuções)
CC-LH.LH.HH.HH	Utiliza o playload para gerar as cargas externas e a cada 4 processadores tem-se: dois processadores com cargas de *traces* do tipo LH e os outros dois com HH.	*Termions* (20 execuções) / *Ray Tracing* (20 execuções)

Tabela 1. Ambientes de execução

da maioria das estratégias mestre-escravo, com ganhos entre 0,17% e 10,45%. Na aplicação *Ray Tracing*, o Est e o EstNWS não apresentaram um bom desempenho para a imagem *room*. A maioria das estratégias mestre-escravo (com exceção da MEBF1 com 16 processadores) obteve ganhos entre 7% e 41,8% em relação às estratégias estáticas. Este fato pode ser explicado porque a imagem *room* é bastante heterogênea em relação ao tempo de computação de cada *pixel*. Embora mais heterogênea que a imagem *5balls*, a renderização da imagem *room* é 80% mais rápida, pois a renderização da imagem *5balls* exige cálculos mais completos para a maioria dos seus *pixels*. As estratégias do tipo mestre-escravo, para as duas aplicações, apresentaram desempenhos semelhantes. Nos *Termions*, apenas nas execuções com 4 processadores foi possível observar diferenças de desempenho. Esse comportamento pode ser justificado pelos desbalanceamentos gerados na distribuição do último bloco de tarefas. O mesmo ocorreu no *Ray Tracing*. Para a imagem *5balls* com 16 processadores observa-se que a MEBF8 apresentou o pior desempenho, justificado por um grande desbalanceamento (IDC = 21,88%). Isso se deve ao fato de que muitas vezes o mestre envia o último bloco de tarefas para um escravo enquanto que os demais ficam ociosos até que este termine de executar as tarefas recebidas. Para a imagem *room*, a estratégia MEBF1 foi a que apresentou o pior desempenho, que pode ser explicado pela heterogeneidade das tarefas, pelo maior *overhead* de comunicação com o envio de blocos unitários de tarefas, e pelo menor tempo para renderizar essa imagem.

Os experimentos realizados no ambiente CC-0.1.2.3 (Figuras 2 e 6) mostraram o problema de se executar a estratégia estática simples em um ambiente com a presença de carga externa. Observa-se que esta estratégia apresentou grandes desequilíbrios de carga. A estratégia EstNWS foi executada para tentar amenizar este problema. Com os *Termions*, embora o tempo total de computação tenha sido melhor do que o estático simples (para 4, 8 e 16 processadores ganhos de 37,6%, 97,1% e 98,1%), as execuções com 4 processadores apresentaram em média 25% de desbalanceamento com 4 processadores, o que mostra que essa

estratégia não apresentou um bom desempenho quando executada com poucos processadores. Nas execuções com 8 e 16 processadores, o EstNWS obteve desempenho equivalente ao das melhores estratégias do tipo mestre-escravo. Na aplicação *Ray Tracing*, o EstNWS apresentou um bom desempenho para a imagem *5balls* e um desempenho regular para a imagem *room* (IDC médio de 10,15% para a imagem *5balls* e de 40,34% para a *room*).Essa diferença pode ser explicada pela característica mais heterogênea da imagem *room*. A maioria das estratégias mestre-escravo obteve desempenhos semelhantes e eficientes. Apenas a MERB1 com os *Termions* e MERB1 e MEBF8 (com 16 processadores) com a imagem *5balls* não apresentaram um bom desempenho.

No ambiente CC-LL.LL.HL.HL (Figuras 3 e 7), para os *Termions* e na renderização da imagem *5balls*, as estratégias MEBF8 e MERB1 não apresentaram um bom desempenho e observou-se uma piora no desempenho da EstNWS quando comparada com a Est e com a maioria das estratégias mestre-escravo (comparação com os resultados obtidos na execução CC-0.1.2.3). As pequenas variações em processadores com cargas pesadas parecem ter prejudicado a estratégia EstNWS. No caso da imagem *5balls*, a EstNWS apresentou um desempenho pior do que a Est nas execuções com 16 processadores, com uma diferença de 14,07%. Na execução dos *Termions* com 16 processadores observou-se uma diferença significativa entre as estratégias MERB2 e MERB3 e as melhores estratégias MERB (de aproximadamente 44%). Isso porque essas duas estratégias não efetuam recálculo de peso durante a execução e as pequenas variações das cargas geradas pelo *Playload* podem ter tido um maior impacto nas execuções com 16 processadores, pois o tempo total de computação é menor.

Nos experimentos no ambiente CC-LH.LH.HH.HH (Figuras 4 e 8), a aplicação dos *Termions* e a renderização da imagem *5balls* tiveram desempenhos semelhantes. A estratégia EstNWS apresentou o seu pior desempenho em comparação com as estratégias do tipo mestre-escravo. Percebe-se que a diferença entre essa estratégia e a Est diminuiu ainda mais, o que indica que essa estratégia não conseguiu se adaptar bem a um ambiente onde as cargas apresentam uma alta variação. As estratégias MEBF8, MERB1, MERB2 e MERB3 também não apresentaram um bom desempenho. A MEBF8 apresentou desequilíbrios relevantes nas execuções com 8 e 16 processadores (nos *Termions*, IDC = 4,62% para 8 processadores e IDC = 10% para 16, e no *Ray Tracing*, IDC = 6,95% para 8 e IDC = 22,22% para 16). A MERB1 novamente não apresentou um bom desempenho. Para a imagem *5balls*, a MERB2 e a MERB3 foram fortemente prejudicadas nas execuções com 4 processadores devido às grandes variações das cargas externas nesse ambiente. No caso da estratégia MERB3, o fato de o peso ser calculado apenas no início pode ter prejudicado essa estratégia, por não ter refletido as variações de carga. Embora a estratégia MERB4 também calcule os pesos apenas no início da execução, o uso da disponibilidade de CPU, fornecida pelo NWS, favoreceu essa estratégia. Com os *Termions*, essa situação só pode ser observada na execução com 16 processadores. Observa-se uma piora no desempenho da estratégia MERB6 com 16 processadores na aplicação dos *Termions*. O fato do TMCT refletir uma situação verificada no passado pode ter prejudicado essa estratégia, pois a informação recebida se tornou desatualizada devido à alta variação das cargas nesse ambiente. Dessa forma, o peso calculado não estaria refletindo a capacidade real de processamento de um escravo, o que gerou desbalanceamentos (IDC = 15,09%). Além desse fator, o *overhead* gasto no recálculo dos pesos a cada fim de lote também pode ter contribuído para uma queda no desempenho.

No ambiente CC-LH.LH.HH.HH, percebe-se, para ambas as aplicações, nas execuções com 8 e 16 processadores, uma perda de escalabilidade nas execuções das estratégias estáticas, pois, mesmo com o dobro de processadores, os tempos de execução foram bem mais próximos. Além disso, pode-se notar que a estratégia EstNWS obteve um desempenho pior na renderização da imagem *room*, principalmente com 4 processadores, quando obteve o pior desempenho entre todas as estratégias.

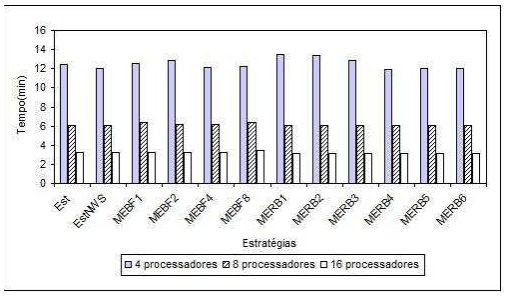

Figura 1. Termions – SC

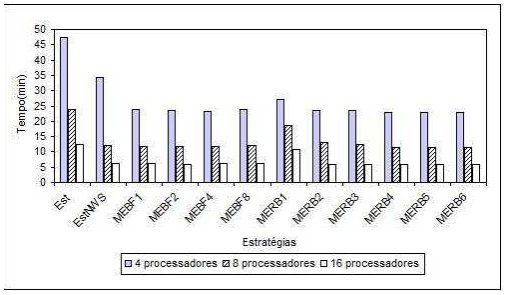

Figura 2. Termions – CC-0.1.2.3

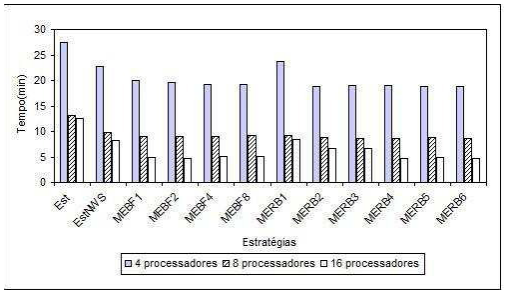

Figura 3. Termions – CC-LL.LL.HL.HL

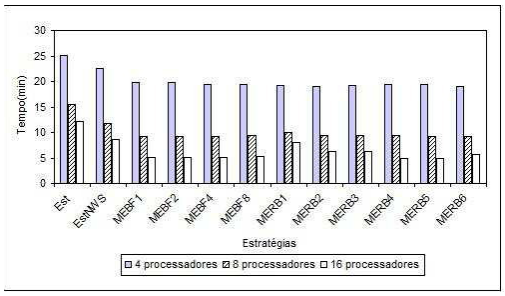

Figura 4. Termions – CC-LH.LH.HH.HH

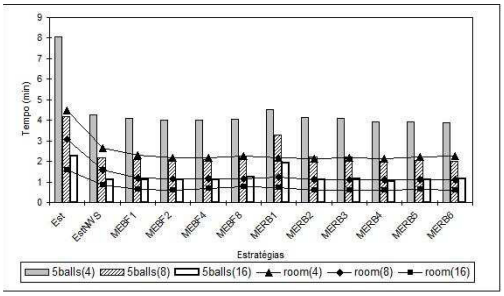

Figura 6. Ray Tracing – CC-0.1.2.3

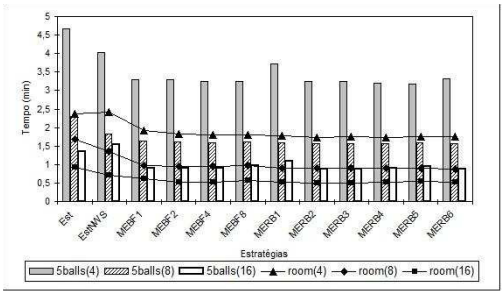

Figura 7. Ray Tracing – CC-LL.LL.HL.HL

5.2. Experimentos Realizados no Grid

Nesta seção, serão apresentados os principais resultados dos experimentos realizados em um *grid* computacional. Análises complementares podem ser encontradas em [3].

As estratégias foram executadas em um *grid* formado por dois *clusters* assim identificados: *cluster*PUC (*cluster* da PUC-RIO) e *cluster*UFF (*cluster* da UFF-Niterói). O *grid* era formado por 30 máquinas, sendo 15 no *cluster*PUC, com processadores Pentium II (398 MHz) e 15 no *cluster*UFF, com processadores Pentium IV (2.65 GHz). As máquinas em cada *cluster* eram conectadas por uma rede *Fast Ethernet* e os dois *clusters* interconectados via Internet. Para possibilitar a comunicação entre os processadores dos dois *clusters* foi utilizado o *middleware* Globus e a biblioteca de troca de mensagens *mpilam*.

Além das estratégias avaliadas no *cluster*, foram executadas a estratégia MEH e uma variação da estratégia MERB6 que também utiliza a latência de rede (monitorada pelo NWS) como parâmetro auxiliar na definição do peso de um processador. A versão original será denominada MERB6a e essa nova versão, MERB6b. É importante ressaltar que não serão apresentados os resultados das estratégias Est, MERB1 e MERB2, pois apresentaram um desempenho inferior.

Na estratégia MEH, o tamanho do bloco de tarefas que o submestre solicita para o mestre global ficou assim definido: 45 tarefas na aplicação dos *termions* (3 para cada escravo),

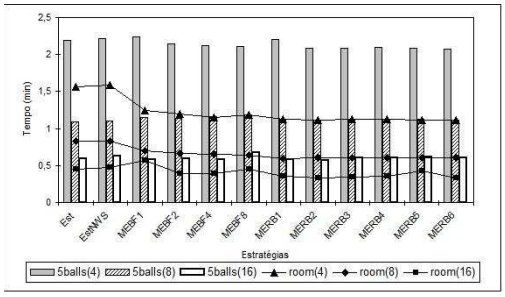

Figura 5. Ray Tracing – SC

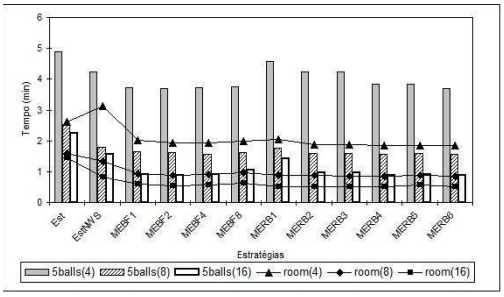

Figura 8. Ray Tracing – CC-LH.LH.HH.HH

30 tarefas no *Ray Tracing* (2 para cada escravo) e 105 tarefas na aplicação sintética (7 para cada escravo).

A maior quantidade de processadores no *grid* exigiu aumentar o tempo de computação das aplicações avaliadas. A aplicação dos *termions* foi executada com 2000 tarefas. A aplicação *Ray Tracing* utilizou a imagem *5balls*, que leva mais tempo para ser renderizada, e cada *pixel* foi dividido em 64 sub-regiões para aumentar o tempo de computação de cada um deles. A aplicação sintética foi executada com 20000 tarefas cujo tempo de computação foi configurado para durar aproximadamente 2s no processador mais lento do *grid*(processadores do *cluster*PUC).

Os experimentos foram realizados com duas configurações de rede no *grid*. Na primeira, a comunicação inter-*cluster* (entre máquinas em *cluster* distintos) utilizou a latência real de comunicação entre os *clusters* (entre 1ms e 4ms). A outra configuração utilizou a ferramenta TC para emular uma rede com canais de comunicação mais lentos entre os *clusters*. A latência inter-*cluster* foi configurada com o valor de 250ms.

Inicialmente, todas as aplicações foram executadas com o conjunto inicial de tarefas localizado no *cluster*UFF. Nas estratégias mestre-escravo, o processo mestre foi executado na máquina do *cluster* que possuía o conjunto inicial de tarefas (na estratégia MEH o mestre global foi executado nessa máquina e o submestre em uma máquina do outro *cluster*). O conjunto de estratégias foi executado cinco vezes e os tempos fornecidos são uma média dos tempos.

Nas duas aplicações reais avaliadas (*Termions* e *Ray Tracing*), observaram-se resultados semelhantes quando o conjunto inicial de tarefas estava localizado no *cluster*PUC. Esses resultados podem ser observados nas Figuras 9 e 10. Nos dois casos, observa-se que quando a latência inter-*cluster* foi de 250ms (gráficos de linhas), a maioria das estratégias apresentaram um desempenho melhor do que quando a latência inter-*cluster* foi a real (gráficos de barras). Na aplicação dos *Termions*, foi possível observar duas exceções a esta observação, pois as estratégias MEBF1 e MEBF8 apresentaram um desempenho melhor quando a latência inter-*cluster* foi a real, o que era o esperado. Na MEBF1 o motivo foi o alto *overhead* de comunicação gerado pelo envio unitário de tarefas para as máquinas do *cluster*UFF através de um canal lento e, na MEBF8, pelo alto índice de desbalanceamento (alto IDC – 26,7%) gerado por uma máquina do *cluster*PUC que, possivelmente, tenha recebido o último bloco de tarefas.

Observa-se que, mesmo com o atraso no envio de tarefas para o *cluster*UFF, forçando o *cluster*PUC a executar mais tarefas, os experimentos com a latência inter-*cluster* de 250ms obtiveram um desempenho melhor. Nos *Termions*, quando comparadas as duas configurações de latência, com 250ms os ganhos foram, em média, de 7% com percentuais variando entre 0,6% e 15%. No *Ray Tracing*, a diferença foi ainda maior sendo, em média, 26,61% mais rápido com

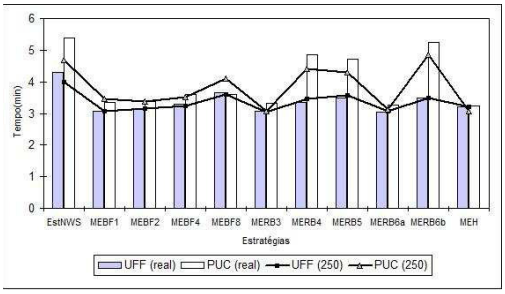

Figura 9. Termions

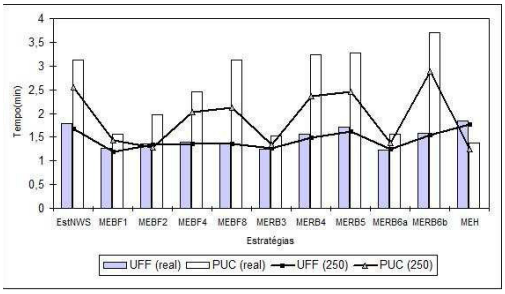

Figura 10. Ray Tracing

250ms, variando entre 8% e 54,2%. O motivo desse desempenho ruim com a latência mais baixa pode ser explicado pelo baixo potencial de processamento do mestre (localizado no *cluster*PUC) e, consequentemente, pela sua capacidade em atender prontamente os seus escravos. Nota-se a ocorrência de um "gargalo" no mestre, que acabou sub-utilizando as máquinas do *cluster*PUC, uma vez que o número excessivo de requisições feitas por máquinas do *cluster*UFF, que executam tarefas mais rapidamente, atrasaram o envio de tarefas localmente.

Na aplicação Sintética, o comportamento foi diferente (Figura 11), pois quando o mestre estava localizado no *cluster*PUC o desempenho foi, na maioria das vezes, melhor no *grid* com latência real (com exceção das estratégias EstNWS, MERB4 e MERB5). Os ganhos foram, em média, de 25,45% com percentuais variando entre 2,2% e 48%. Nesse caso, o maior número de tarefas e o maior tamanho delas foi determinante para esse comportamento. O envio de tarefas de tamanho maior para as máquinas do *cluster*UFF embutiram um maior *overhead* no tempo total de computação e foram mais significativos do que um possível "gargalo" no mestre.

6. Conclusões

Neste trabalho, analisou-se o desempenho de diversas estratégias de balanceamento de carga, utilizando-se

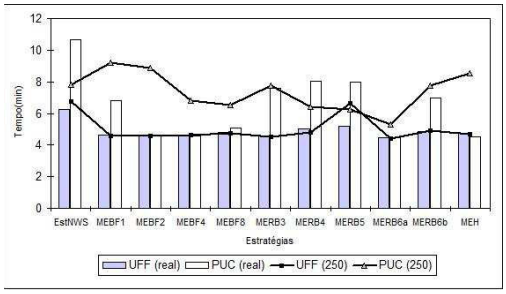

Figura 11. Sintético

aplicações distintas, em *clusters* e *grids* computacionais.

A esperada necessidade de utilização de estratégias dinâmicas se confirmou após a análise das estratégias estáticas. O estático simples se mostrou bastante sensível à heterogeneidade de processadores e à presença de carga externa. Embora a estratégia estática com NWS tenha amenizado o impacto gerado por esses fatores, ainda apresentou altos índices de desbalanceamento, principalmente em ambientes que apresentam variação de carga externa ou que possuem recursos computacionais heterogêneos.

As diferentes implementações das estratégias do tipo mestre-escravo se mostraram, na maioria das vezes, alternativas interessantes de algoritmos dinâmicos de balanceamento de carga. O problema da definição do tamanho ideal do bloco de tarefas enviadas pelo mestre aos seus escravos foi constatado após a análise dos resultados computacionais. Observou-se que as estratégias com blocos de tamanho fixo pequenos podem ser prejudicadas pelo aumento do *overhead* de comunicação, principalmente quando as tarefas possuírem um tamanho grande. Em contrapartida, blocos de tamanho grande tenderam a prejudicar o desempenho devido ao desbalanceamento provocado após o envio dos últimos blocos.

As estratégias do tipo *weighted factoring* apresentaram bons desempenhos. No *cluster*, o desempenho dessas estratégias se mostrou sensível aos cenários onde as cargas externas variavam com maior frequência e, no *grid*, o desempenho se apresentou dependente da heterogeneidade dos escravos e das condições dos canais de comunicação.

A realização de experimentos com diferentes aplicações SPMD mostrou que o desempenho das estratégias também é influenciado pelas características da aplicação, tais como o tamanho, a heterogeneidade e a quantidade de tarefas. Além disso, os diferentes ambientes e condições de execução também evidenciaram que o desempenho das estratégias está relacionado às características da arquitetura paralela adotada.

Referências

[1] AINDA, K., FUTAKATA, Y., e HARA S. High-Performance Parallel and Distributed Computing for the BMI Eigenvalue Problem. *Proc. of the 16th IPDPS*, pp. 71-78, 2002.

[2] ARGOLO, G., OLIVEIRA, A., MARTINS, S. et al. Avaliação de Estratégias de Balanceamento de Carga do tipo Mestre-Escravo em um Grid Computacional. *Workshop de Grades Computacionais e Aplicações*, Petrópolis, 2004.

[3] OLIVEIRA, A. Avaliação de Estratégias de Balanceamento de Carga do Tipo Mestre-Escravo para Aplicações SPMD em *Clusters* e *Grids* Computacionais. *Dissertação de Mestrado, COPPE Sistemas, UFRJ*, Brasil, 2008.

[4] BERMAN, F., SHAO, G. e WOLSKI, R. Master/Slave Computing on Grid. *Proc. of the 9th Heterogeneous Computing Workshop*, pp. 3-16, Cancun, México, 2000.

[5] COOK, R.L., PORTER, T. e CARPENTER, L. Distributed Ray Tracing. *ACM SIGGRAPH Computer Graphics*, Vol. 18(4), pp 165-174, Julho/1984.

[6] DINDA, P. e HALLARON, D. O. Realistic CPU Workloads Through Host Load Trace Playback. *Proc. of the 5th Workshop on Languages, Compilers and Run-time Systems for Scalable Computers, LNCS*, pp. 246-259, Rochester, New York, Maio/2000.

[7] FOSTER, I. e KESSELMAN C. Globus: A metacomputing infrastructure toolkit. *The Intl. Journal of Supercomputer Applications and High Performance Computing*, Vol 11, pp. 115-128, 1997.

[8] GOUX, J. P., KULKARNI, S., YODER, M. et al. An Enabling Framework for Master-Worker Applications on the Computational Grid. *Proc. of HPDC, IEEE Computer Society Press*, pp. 43-50, Pittsburgh, Pennsylvania, Aug/2000.

[9] HUMMEL, S. F., SCHONBERG, E. e FLYNN, L.E. Factoring: A Method for Scheduling Parallel Loops. *Communications of the ACM*, Vol. 35, N.8, pp. 90-101, Agosto/1992.

[10] HUMMEL, S. F., SCHMIDT, J., Uma, R.N., e WEIN, J. Load-Sharing in Heterogeneous Systems via Weighted Factoring. *Proc. of the 8th Annual ACM Symposium on Parallel Algorithms and Architectures*, SPA, pp. 318-328, 1996.

[11] Página Web contendo os *Traces* do *Playload* utilizados neste trabalho. http://cs.uchicago.edu/lyang/load (última consulta realizada em Maio/2008).

[12] PLASTINO, A., COSTA, R., THOME, V., et al. Exploring Load Balancing in a Scientific SPMD Parallel Application. *4th International Workshop on HPSECA/ICPP*, 2002.

[13] PLASTINO A., RIBEIRO, C.C. e RODRIGUEZ, N. Development SPMD Applications with Load Balance. *Parallel Computing*, vol. 29, pp. 743-766, 2003.

[14] PLASTINO, A., THOMÉ V., VIANNA, D. et al. Load Balancing in SPMD Applications: Concepts and Experiments. *High Performance Scientific and Engineering Computing: Hardware/Software Support*, Kluwer Academic Publishers, pp. 95-107, 2004.

[15] POLYCHRONOPOULOS, C. D. e KUCK, D. J. Guided Self-Scheduling: A Pratical Scheduling Scheme for Parallel Supercomputers. *IEEE Transaction on Computers*, Vol. C-36, N.12, pp. 1425-1439, Dezembro/1987.

[16] SILVEIRA, O. T. Dispersão Térmica em Meios Porosos Periódicos. Um Estudo Numérico. *Tese de Doutorado, Instituto Politécnico - UERJ*, Brasil, 2001.

[17] WHITTED, T. An Improvement Illumination Model for Shaded Display. *Communications of the ACM*, Vol.23(6), pp. 343-349, Junho/1980.