ComprehensiveBench: Um Benchmark Flexível para Avaliação de Balanceadores de Carga no Ambiente de Programação Charm++

Tiago C. Bozzetti[1], **Laércio L. Pilla**[2],
Márcio Castro[2], **Philippe O. A. Navaux**[1]

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

{tcbozzetti,navaux}@inf.ufrgs.br

[2]Departamento de Informática e Estatística – Centro Tecnológico
Universidade Federal de Santa Catarina (UFSC)
Caixa Postal 476 – 88.040-900 – Florianópolis – SC – Brasil

{laercio.pilla,marcio.castro}@ufsc.br

Abstract. *Parallel applications that feature imbalanced task loads usually do not exploit the full potential of the underlying hardware resources. In this context, load balancing techniques rise as a way to minimize this problem, and consequently, to improve the performance of parallel applications. However, identifying the optimal load balancing algorithm for a specific context is not a trivial task. In this context, this paper presents a new benchmark to evaluate periodic load balancers implemented in Charm++. It can be used to assist users in the development of new load balancing techniques as well as to help users to choose a specific load balancer for an arbitrary application. Finally, this paper presents a performance evaluation of different periodic load balancers available with Charm++ using the proposed benchmark.*

Resumo. *Em aplicações paralelas que possuem tarefas com cargas desbalanceadas, os recursos de hardware não são utilizados eficientemente. Técnicas de balanceamento de carga visam amenizar esse problema e, consequentemente, podem melhorar o desempenho da aplicação paralela. Entretanto, identificar qual algoritmo de balanceamento de carga seria o mais indicado em um dado contexto é um problema não trivial. Nesse contexto, este artigo apresenta um benchmark para a avaliação de balanceadores de carga periódicos no ambiente paralelo Charm++. O benchmark tem os objetivos de auxiliar no desenvolvimento de novas técnicas de balanceamento e de auxiliar na escolha de um balanceador de carga específico para uma aplicação paralela qualquer. Por fim, o artigo apresenta avaliações de diferentes balanceadores de carga periódicos do Charm++ realizadas com o benchmark desenvolvido.*

1. Introdução

As tarefas que compõem uma aplicação paralela podem apresentar cargas de trabalho distintas. Esse desbalanceamento de carga não gera perdas no desempenho da aplicação se for mínimo. Porém, um desbalanceamento de carga mais acentuado acarreta em um uso

ineficiente dos recursos de hardware e, portanto, a aplicação apresentará um desempenho inferior. Nesse contexto, a função de algoritmos de balanceamento de carga é a de controlar a distribuição das tarefas que compõe a aplicação entre os núcleos de processamento do sistema paralelo. Eles utilizam o passado recente (últimas iterações) da aplicação como preditor do futuro próximo (próximas iterações). Dessa forma, é possível minimizar o desbalanceamento de carga para um uso mais eficiente das unidades de processamento. Além disso, balanceadores de carga podem considerar a topologia da máquina e o perfil de comunicação da aplicação para minimizar os custos de comunicação entre as tarefas [Pilla et al. 2012, Pilla et al. 2014]. Os algoritmos empregados nos balanceadores de carga são baseados em heurísticas, dado que o problema de balanceamento de carga é NP-Completo [Leung 2004].

Para que o balanceador de carga tome a decisão mais adequada, diversas informações da aplicação e da máquina podem ser utilizadas. Ao considerar a topologia da máquina e o perfil de comunicação da aplicação, por exemplo, o balanceador de carga pode ser eficaz em aplicações em que as tarefas se comunicam com mais intensidade em uma plataforma hierárquica, onde os tempos de comunicação entre as diferentes unidades de processamento variam. O balanceador de carga pode mapear as tarefas que possuem uma comunicação mais custosa para nós próximos na hierarquia. Além das informações da máquina e da aplicação utilizadas pelos balanceadores, outras características podem determinar se um balanceador de carga dinâmico será eficaz ou não para uma dada aplicação, tais como o número de migrações executadas e a complexidade do algoritmo empregado. Portanto, é fundamental simular aplicações paralelas em diferentes contextos e identificar propriedades dos balanceadores de carga para a tomada da melhor escolha possível.

Os *benchmarks* atuais utilizados para avaliar balanceadores de carga ignoram ou limitam algumas características de aplicações paralelas que são determinantes para o desempenho dos balanceadores. O volume de dados associado às tarefas, que influi diretamente no custo das migrações executadas pelos balanceadores de carga, não pode ser especificado nos *benchmarks* atuais. Outras características ainda mais significativas como a carga computacional e o perfil de comunicação das tarefas são simuladas com o mínimo de dinamicidade nos *benchmarks* mais expressivos, o que não reflete a realidade de uma série de aplicações científicas que apresentam cargas de trabalho irregulares e dinâmicas. Dado o contexto apresentado, este artigo tem por objetivo propor um *benchmark* chamado COMPREHENSIVEBENCH[1] para a avaliação de balanceadores de carga que considera todas as propriedades determinantes de aplicações paralelas e que é expressivo na forma em como essas propriedades são especificadas para que seja possível simular aplicações com características dinâmicas.

O restante desse artigo é organizado da seguinte forma: o *benchmark* de balanceamento de carga desenvolvido é descrito na Seção 2. O seu uso para a avaliação de algoritmos de balanceamento de carga é discutido na Seção 3. Por fim, as conclusões finais e trabalhos futuros são apresentados na Seção 4.

[1]Disponível no repositório do projeto HieSchella: http://forge.imag.fr/projects/hieschella/

2. COMPREHENSIVEBENCH

A proposta de um novo *benchmark* para a avaliação de algoritmos de balanceamento de carga é embasada na falta de flexibilidade das aplicações usadas para esse fim atualmente. Exemplos de avaliações envolvendo balanceadores de carga encontrados na literatura [Rodrigues et al. 2010, Zheng et al. 2011, Menon et al. 2012, Pilla et al. 2012, Pilla et al. 2014, Tesser et al. 2014] usam *benchmarks* como: *lb_test* e *stencil4D*, os quais focam em uma variação de carga estática entre tarefas e padrões bem definidos de comunicação; *kNeighbor*, o qual estressa problemas de comunicação apenas; e *LeanMD* e *Ondes3D*, os quais são aplicações reais. Em todos os casos listados, há uma falta de flexibilidade na definição da carga das tarefas e na quantidade de memória utilizada pelas tarefas. Essas limitações guiam o desenvolvimento e implementação do *benchmark* COMPREHENSIVEBENCH.

Para a implementação de COMPREHENSIVEBENCH, a linguagem Charm++ [Kale and Krishnan 1993] foi utilizada. Charm++ é uma extensão de C++ que permite a instanciação de objetos que se comunicam assincronamente em um ambiente paralelo. Charm++ também conta com um *framework* de balanceamento de carga [Zheng et al. 2011], o qual inclui uma série de balanceadores de carga que podem ser avaliados pelo *benchmark* proposto, o que justifica seu uso. As aplicações em Charm++ são compostas por objetos denominados *chares* que se comunicam por meio de mensagens assíncronas. O ambiente de execução do Charm++ gerencia as mensagens enviadas pelos *chares* e os recursos físicos da plataforma, além de coletar as informações da aplicação para fornecer aos balanceadores de carga. A partir das informações coletadas pelo *runtime* e do mapeamento corrente das tarefas, o balanceador pode definir um novo mapeamento para que o runtime realize as migrações de tarefas necessárias. Algumas limitações dos balanceadores de carga do Charm++ são decorrentes das limitações do *framework* de balanceamento de carga que não fornece todas as informações que são determinantes para o desempenho dos balanceadores.

Para uma avaliação eficaz de balanceadores de carga, o *benchmark* considera uma série de propriedades de aplicações paralelas. A Tabela 1 contém as principais propriedades de aplicações paralelas consideradas.

A execução do *benchmark* é dividida em iterações. Em cada iteração, as tarefas executam uma série de operações, que correspondem à carga computacional. Após a execução das operações, as tarefas enviam mensagens para as tarefas vizinhas conforme o grafo de comunicação utilizado e então a próxima iteração é iniciada. A carga computacional das tarefas, o número de mensagens e o tamanho das mensagens enviadas para as tarefas vizinhas podem variar entre as iterações. Dessa forma, o *benchmark* é capaz de simular o comportamento dinâmico de aplicações paralelas. Da mesma forma que as propriedades variam ao longo das iterações, essas propriedades podem assumir valores diferentes em tarefas diferentes. Essa característica do *benchmark* permite simular uma aplicação com uma distribuição de carga irregular entre as tarefas, tornando necessário o uso de uma estratégia de balanceamento de carga. O restante das propriedades são estáticas, ou seja, não variam ao longo das iterações.

As propriedades da aplicação paralela simulada pelo *benchmark* são definidas por meio de expressões matemáticas, podendo conter constantes, operações e variáveis. Esse mecanismo provê uma flexibilidade superior à encontrada em outros *benchmarks*. As ex-

Tabela 1. Propriedades consideradas por COMPREHENSIVEBENCH e expressões correspondentes.

Propriedade	Expressão
Número de unidades de processamento	p
Número de tarefas	n
Número de iterações	r
Frequência de chamada do balanceador	lbfreq
Mapeamento inicial da tarefa	initmap
Grafo de comunicação	gcomm
Volume de dados da tarefa	tasksize
Carga computacional da tarefa	load
Número de mensagens trocadas	msgnum
Tamanho das mensagens	msgsize

pressões seguem a sintaxe da linguagem C++ e devem obrigatoriamente resultar em um valor numérico para que sejam corretamente avaliadas pelo *benchmark*. Para diferenciar cada expressão de acordo com a propriedade que está sendo definida, é designado um nome para cada expressão. Ao modificar os valores das variáveis inseridas na expressão, a sua avaliação também pode resultar em um valor modificado. As variáveis inseridas nas expressões podem representar o identificador da tarefa ou o número da iteração. Isso permite que propriedades assumam valores distintos quanto são avaliadas por tarefas distintas e em iterações diferentes. As expressões que definem as propriedades estáticas são avaliadas no início da execução do *benchmark*, enquanto as expressões de propriedades dinâmicas são avaliadas em vários momentos durante a execução.

COMPREHENSIVEBENCH, assim como todo programa escrito em Charm++, é composto por objetos chamados *chares*. Os *chares* simulam o comportamento das tarefas que compõem a aplicação paralela. No início da execução, os *chares* são mapeados para as unidades de processamento disponíveis. O *benchmark* permite definir qualquer estratégia de mapeamento inicial através da expressão initmap, podendo assim criar uma situação de desbalanceamento de carga no início da execução. A Expressão 1 define o mapeamento inicial das tarefas em razão do identificador da tarefa (task_id) e do número de unidades de processamento disponíveis (p) no estilo Round-Robin. O mapeamento tradicional das aplicações em Charm++ é realizado em blocos, podendo ser definido pela Expressão 2. É possível alocar todas as tarefas em uma única unidade de processamento, como na Expressão 3, em que todas as tarefas são inicialmente mapeadas para a unidade com o identificador igual a 2.

```
initmap = task_id % p                (1)
initmap = (task_id / p) % p          (2)
initmap = 2                          (3)
```

Cada *chare* possui uma quantidade arbitrária de dados associada, incluindo o código do *chare* e as variáveis que ele manipula, que deve ser transferida de uma unidade

de processamento para outra quando a tarefa é migrada por um balanceador de carga. Esse volume de dados influi diretamente no custo da migração e é definido pela expressão tasksize. A Expressão 4 define que todas as tarefas que compõem a aplicação possuem 100 bytes. Já a Expressão 5 é mais complexa e utiliza uma função exponencial para definir o volume de dados de cada tarefa. Dessa forma, o custo de migração das tarefas com os menores identificadores é inferior ao custo de migração das tarefas com maiores identificadores. Um balanceador de carga, ao utilizar a informação do tamanho das tarefas, pode optar por migrar ou não migrar uma determinada tarefa de acordo com um custo estimado para sua migração.

tasksize = 100	(4)
tasksize = 100 + task_id $*$ task_id	(5)

Em cada iteração, os *chares* executam uma série de operações e em seguida trocam mensagens com seus vizinhos. A série de operações realizadas simulam a carga computacional das tarefas que é definida pela expressão load. Essa expressão é avaliada no início de cada iteração para que seja possível alterar a carga de trabalho das tarefas durante a execução, ou seja, simular uma carga de trabalho dinâmica. As operações realizadas pelos *chares* são necessárias para que o *framework* de balanceamento de carga do Charm++ colete as informações da carga computacional dos *chares* e as forneça aos balanceadores de carga. Para obter uma maior precisão na especificação da carga computacional, a avaliação da expressão não determina o número de operações a serem executadas, mas o tempo em milissegundos que essas operações devem durar. Pode-se inserir na expressão load variáveis que representam o identificador da tarefa (task_id) e o número da iteração (iteration_number). A Expressão 6 define uma aplicação com uma distribuição de carga irregular e dinâmica. A aplicação gerada por essa expressão possui tarefas com cargas irregulares no início da execução e a diferença de carga entre as tarefas cresce conforme as iterações avançam. O operador condicional pode ser utilizado para dividir as tarefas em dois grupos com base no número total de tarefas (n) e atribuir a cada um deles uma carga computacional diferente, o que é realizado na Expressão 7.

load = 2 $*$ iteration_number $*$ task_id	(6)
load = task_id $<$ n/2 ? 10 : 50	(7)

Os vizinhos de cada *chare* são definidos de acordo com a topologia de comunicação definida através da expressão gcomm. As topologias de comunicação disponíveis no momento são anel, malha de duas dimensões e malha de três dimensões. A Expressão 8 define uma topologia de comunicação em anel. Os *chares* sabem quantos e quem são seus vizinhos no início da execução do *benchmark*. Dessa forma, o *chare* é capaz de terminar a iteração após enviar e receber as mensagens de todos seus vizinhos. O número de mensagens que um *chare* envia para cada vizinho e o tamanho dessas mensagens são definidas através das expressões msgnum e msgsize, que, assim como a expressão load, podem utilizar variáveis que representam o identificador da tarefa e o número da iteração e são avaliadas antes do início de cada iteração. O número de mensagens enviadas, ao se utilizar a Expressão 9, é dinâmico, pois incrementa em um após cada sequência de 5 iterações. Na Expressão 10, o operador condicional é utilizado

para dividir o número total de iterações (r) em dois intervalos, atribuindo um tamanho de 50 bytes para as mensagens enviadas na primeira metade e um tamanho de 10.000 bytes para as mensagens enviadas na segunda metade das iterações.

gcomm = Ring	(8)
msgnum = 1 + iteration_number/5	(9)
msgsize = iteration_number $<$ r/2 ? 50 : 10000	(10)

O balanceador de carga que será utilizado é chamado pela aplicação entre intervalos de iterações fixos especificados pela expressão lbfreq. A Expressão 11 define um intervalo de 5 iterações entre cada chamada do balanceador de carga. Ao definir a frequência de chamada do balanceador de carga é necessário considerar o tempo de execução do balanceador, o que pode não compensar o seu ganho de desempenho.

lbfreq = 5	(11)

3. Avaliação experimental

COMPREHENSIVEBENCH foi utilizado para avaliar diferentes balanceadores de carga fornecidos por Charm++. Os resultados obtidos com os balanceadores foram comparados com execuções sem balanceamento de carga. A máquina utilizada para a avaliação é composta por 2 processadores Intel(R) Xeon(R) E5530 cadenciados à 2.40 GHz, possuindo 4 núcleos por processador e 24 GB de memória RAM. Os resultados apresentados representam a média de 10 execuções, com desvio padrão máximo de 1.56%.

Primeiramente será feita uma breve descrição dos balanceadores de carga utilizados. Em seguida, serão apresentados os resultados obtidos com cada um deles.

3.1. Balanceadores de carga

Neste trabalho foram utilizados cinco balanceadores de carga: GreedyLB, GreedyCommLB, RefineLB e RefineCommLB, RandCentLB. As principais características de cada um desses balanceadores são discutidas a seguir.

GreedyLB: Algoritmo guloso centralizado que utiliza apenas a carga das tarefas para tomar suas decisões [Zheng 2005]. Após ordenar as tarefas em ordem decrescente de acordo com suas cargas, iterativamente mapeia a tarefa com a maior carga para a unidade de processamento com a menor carga. O mapeamento inicial das tarefas é ignorado e, portanto, um grande número de migrações é esperado. É utilizado pera corrigir rapidamente o desbalanceamento de carga.

GreedyCommLB: Similar a GreedyLB, porém considera o perfil de comunicação da aplicação. O algoritmo calcula a carga de comunicação das tarefas de acordo com o volume de dados que elas enviam para tarefas mapeadas em outras unidades de processamento. A carga de comunicação de uma unidade de processamento é a soma das cargas de comunicação das tarefas que estão associadas à ela. Como para GreedyLB, as tarefas são ordenadas em ordem decrescente de acordo com suas cargas. Iterativamente, o algoritmo seleciona a tarefa de maior carga e a mapeia para a unidade de processamento com a menor carga total, que é calculada através da carga computacional e de comunicação da

unidade de processamento. O algoritmo tende a mapear tarefas que possuem um maior custo de comunicação na mesma unidade de processamento.

RefineLB: É um algoritmo menos agressivo se comparado às estratégias gulosas. Considera o mapeamento inicial das tarefas e iterativamente o otimiza até atingir um estado em que nenhuma unidade de processamento está sobrecarregada ou até que nenhuma migração crie um mapeamento onde a carga está mais balanceada. O algoritmo utiliza um limiar para definir se uma unidade de processamento está sobrecarregada. Até atingir um estado de término, verifica todas as possíveis migrações de tarefas da unidade de processamento mais sobrecarregada para todas as unidades que não estão sobrecarregadas e migra a tarefa que torna a sua nova unidade de processamento o mais próximo possível do limiar. Assim como o GreedyLB, não considera o perfil de comunicação da aplicação.

RefineCommLB: Utiliza uma estratégia similar a RefineLB. Ao migrar uma tarefa da unidade de processamento mais sobrecarregada, o algoritmo considera o perfil de comunicação da aplicação para escolher a unidade de processamento destino mais apropriada.

RandCentLB: Não considera nenhuma informação da aplicação. Ignora o mapeamento inicial das tarefas e as mapeia para uma unidade de processamento escolhida aleatoriamente.

3.2. Resultados

Os balanceadores de carga selecionados foram submetidos a diferentes avaliações com o auxílio de COMPREHENSIVEBENCH. O objetivo dessas avaliações é observar o impacto dos balanceadores de carga no desempenho na presença de três fatores: variação do tamanho das tarefas, de cargas estáticas e dinâmicas.

3.2.1. Impacto do tamanho das tarefas

Na primeira avaliação, os balanceadores de carga foram utilizados em aplicações paralelas que apresentam um alto custo de migração e um desbalanceamento de carga ínfimo durante toda a execução. Para gerar um alto custo de migração, as aplicações simuladas possuem em comum um grande volume de dados por tarefa. Dessa forma, é avaliada a capacidade dos diferentes balanceadores de carga em estimar os custos de migração para definir o mapeamento das tarefas.

A Figura 1 apresenta o desempenho dos balanceadores de carga em aplicações paralelas compostas por tarefas de diferentes tamanhos. Foram simuladas 3 aplicações: a primeira utiliza tarefas de 5MB, as tarefas da segunda aplicação possuem 10MB e na terceira as tarefas são de 15MB. As demais propriedades são mantidas constantes. As aplicações executam nas 8 unidades de processamento da máquina, possuem 500 tarefas e executam por 50 iterações. A carga computacional das tarefas é de 10 milissegundos e permanece estática ao longo das iterações. O mapeamento inicial é realizado em blocos, ou seja, a diferença de tarefas associadas entre qualquer par de unidades de processamento é no máximo um. O grafo de comunicação tem topologia de anel e cada tarefa troca uma mensagem de 100 bytes com cada vizinho. O balanceador de carga é chamado a cada 5 iterações. No contexto das três aplicações, o desbalanceamento de carga é

mínimo e, portanto, o sobrecusto de balanceamento de carga não compensa o seu ganho de desempenho.

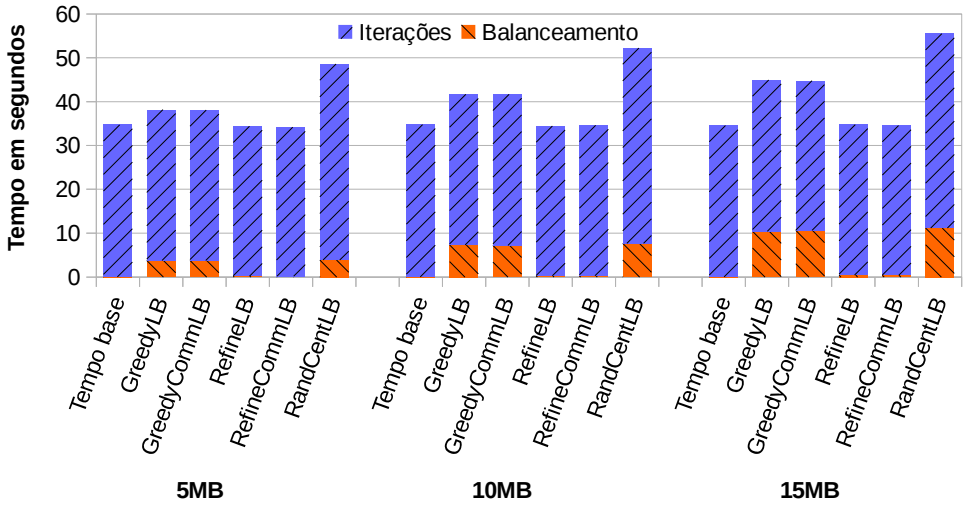

Figura 1. Desempenho dos balanceadores de carga em aplicações com diferentes tamanhos de tarefas.

Os custos de migração de uma tarefa crescem com o seu volume de dados [Pilla et al. 2012]. Considerando as 3 aplicações simuladas, um balanceador de carga que tende a migrar muitas tarefas pode levar a um tempo de execução total da aplicação maior do que outros que não possuem essa característica. Os balanceadores de carga que empregam algoritmos gulosos como GreedyLB e GreedyCommLB, assim como o balanceador aleatório RandCentLB, não consideram o mapeamento das tarefas, o que os leva a realizar um grande número de migrações. Pode-se observar na Figura 1 que esses balanceadores apresentam um sobrecusto de aproximadamente 3.8 segundos na aplicação com tarefas de 5MB. Quando tarefas de 10MB e 15MB são consideradas, tais tempos crescem cerca de 92% e 178%, respectivamente.

Os balanceadores RefineLB e RefineCommLB consideram o mapeamento das tarefas antes de escolher quais migrar. Devido a isso, esses balanceadores, se comparados aos citados anteriormente, tendem a migrar um número pequeno de tarefas e, portanto, apresentam um pequeno sobrecusto nas 3 aplicações simuladas. O desempenho de RefineLB e de RefineCommLB se aproximam da execução sem balanceamento de carga, com um sobrecusto máximo de 0.4 segundos.

3.2.2. Impacto do desbalanceamento

A segunda avaliação utiliza aplicações com volumes de dados por tarefa desprezíveis mas que apresentam distribuições de carga irregulares no inicio de suas execuções. Nas aplicações utilizadas, as tarefas que pertencem a mesma unidade de processamento apresentam a mesma carga de trabalho, enquanto tarefas de unidades de processamento distintas possuem cargas diferentes. Nessa configuração, algumas unidades permanecem em

um estado ocioso aguardando o término do processamento de outras unidades. Com essa avaliação, pode-se observar a velocidade dos balanceadores de carga para mitigar uma situação de desbalanceamento.

Foram simuladas 4 aplicações paralelas com uma distribuição de carga irregular entre as 8 unidades de processamento. As tarefas de cada unidade apresentam a mesma carga de trabalho, porém tarefas de unidades de processamento diferentes possuem possuem cargas distintas. Na primeira aplicação a carga de trabalho das tarefas é proporcional ao identificador da unidade de processamento associada a cada tarefa. Na segunda, a carga de trabalho é proporcional ao dobro do identificador da unidade, na terceira o identificador é multiplicado por 4 e na quarta aplicação é multiplicado por 8. Dessa forma, o desbalanceamento de carga aumenta em intensidade da primeira aplicação até a quarta. A Figura 2 apresenta o desempenho dos balanceadores de carga selecionados nas aplicações desbalanceadas. Cada aplicação utiliza 500 tarefas e 20 iterações. O volume de dados das tarefas é de 100 bytes, o que não representa um custo alto de migração. Assim como na avaliação anterior, o balanceador de carga é chamado a cada 5 iterações. O restante das propriedades também são as mesmas da primeira avaliação.

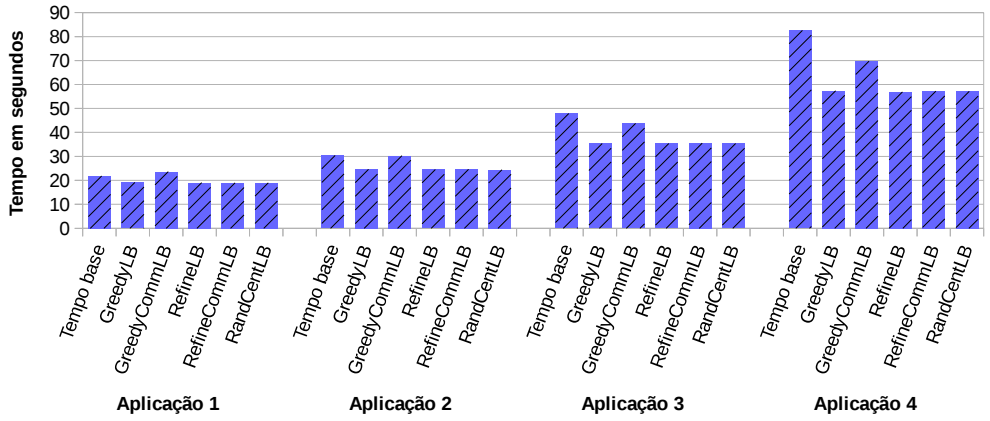

Figura 2. Desempenho dos balanceadores de carga em aplicações desbalanceadas.

A configuração inicial das aplicações determina um estado de desbalanceamento de carga. Pode-se observar na Figura 2 que, ao não utilizar uma estratégia de balanceamento, o desempenho da aplicação é afetado negativamente. Os balanceadores GreedyLB, GreedyCommLB, RefineLB e RefineCommLB conseguem corrigir a situação de desbalanceamento rapidamente. Em relação ao tempo base, eles apresentam um ganho de desempenho de aproximadamente 13% na aplicação de menor desbalanceamento. Já na segunda aplicação o ganho é de 19% e na terceira, com um nível maior de desbalanceamento, o ganho é de 26%. Na aplicação com o maior nível de desbalanceamento, o ganho gerado pelos balanceadores citados é de aproximadamente 31% em relação ao tempo base. A exceção é o balanceador RandCentLB que, devido a sua estratégia aleatória, possui um desempenho próximo à execução sem balanceamento nas aplicações menos desbalanceadas. Porém, na aplicação de maior desbalanceamento, o balanceador apresenta um ganho de desempenho de 15% em relação ao tempo base. Ao contrário da

primeira avaliação, a migração de tarefas não representa um custo alto nessas aplicações e, portanto, todos os balanceadores possuem um sobrecusto desprezível.

3.2.3. Impacto da existência de cargas dinâmicas

A terceira avaliação utiliza aplicações com cargas de trabalho dinâmicas. As aplicações iniciam com um desbalanceamento de carga insignificante e, conforme as iterações avançam, as cargas de trabalho das tarefas são alteradas. Ao utilizar cargas dinâmicas, o balanceamento de carga se torna uma necessidade ao longo da execução. Além disso, as cargas podem retornar a um estado de desbalanceamento mesmo após diversas chamadas de balanceamento de carga.

Nessa avaliação foram simuladas 3 aplicações com cargas de trabalho dinâmicas. As aplicações iniciam a sua execução em um estado em que as cargas estão balanceadas. Assim como na avaliação anterior, as tarefas de uma mesma unidade de processamento apresentam sempre a mesma carga considerando que elas não são migradas. Conforme as iterações avançam, a carga de tarefas de unidades de processamentos diferentes começa a divergir. O grau de afastamento entre as cargas de tarefas de unidades distintas é incrementado na segunda aplicação simulada e novamente incrementado na terceira. Dessa forma, o desbalanceamento gerado pelas cargas de trabalho dinâmicas aumenta em intensidade da primeira aplicação até a terceira. A Figura 3 apresenta o desempenho dos balanceadores de carga selecionados nas aplicações com cargas de trabalho dinâmicas. Assim como na última avaliação, cada aplicação possui 500 tarefas, 20 iterações, o volume de dados das tarefas é de 100 bytes, o balanceador de carga é chamado a cada 5 iterações e o restante das propriedades também são mantidas.

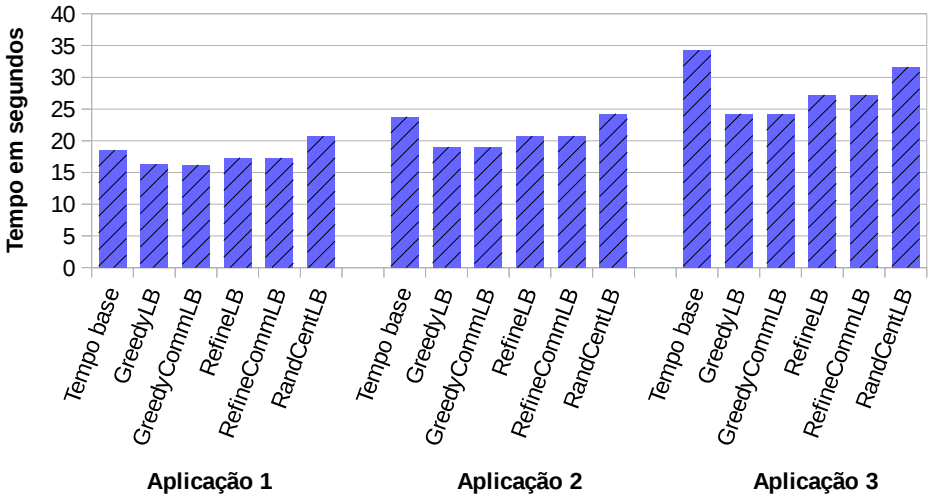

Figura 3. Desempenho dos balanceadores de carga em aplicações com cargas de trabalho dinâmicas.

Como na segunda avaliação, o sobrecusto dos balanceadores pode ser desconsiderado. Na aplicação em que as cargas das tarefas se distanciam mais lentamente, os

balanceadores RefineLB e RefineCommLB apresentam um ganho de aproximadamente 6,7% em relação ao tempo base. Já os balanceadores que empregam algoritmos gulosos, GreedyLB e GreedyCommLB, apresentam um ganho de 12,4%. Essa diferença de desempenho permanece nas outras duas aplicações. Na segunda aplicação, os ganhos de RefineLB e de RefineCommLB são de 12,8% e na terceira são de 20,5%, enquanto os ganhos dos balanceadores gulosos são de cerca de 20,1% e 29,3% em relação ao tempo base. A característica de ignorar o mapeamento anterior torna os balanceadores GreedyLB e GreedyCommLB mais eficazes nesse tipo de aplicação pois são capazes de corrigir mais rapidamente a situação de desbalanceamento. Da mesma forma que na segunda avaliação, o balanceador RandCentLB possui um desempenho próximo à execução sem balanceamento nas aplicações menos desbalanceadas e um pequeno ganho de desempenho na aplicação de maior desbalanceamento. Em relação ao tempo base, esse ganho é de aproximadamente 7,7%.

4. Conclusão

O problema de balanceamento de carga é sabidamente NP-Completo, o que leva ao desenvolvimento de diferentes algoritmos e heurísticas para o aprimoramento do desempenho de aplicações paralelas. A dificuldade na avaliação e comparação desses algoritmos de balanceamento de carga nos levou ao desenvolvimento de COMPREHENSIVEBENCH, um *benchmark* flexível que permite definir características como a carga das tarefas e a quantidade de memória de cada uma delas através de expressões matemáticas, além de permitir a definição de características dinâmicas.

A utilização de COMPREHENSIVEBENCH para a avaliação de balanceadores de carga foi ilustrada em três cenários diferentes, cada qual envolvendo pelo menos três configurações de parâmetros. O primeiro cenário salientou o sobrecusto de migração de tarefas com grandes quantidades de memória por algoritmos que não consideram o mapeamento inicial de tarefas em suas decisões. O segundo cenário demonstrou o uso de balanceadores de carga para uma aplicação estaticamente desbalanceada, enquanto o terceiro cenário tratou de um desbalanceamento dinâmico. Em todos cenários, COMPREHENSIVEBENCH permitiu diferenciar o desempenho dos cinco algoritmos de balanceamento de carga quando possível.

Como trabalhos futuros, nós consideramos a avaliação de algoritmos de balanceamento de carga em plataformas maiores, o desenvolvimento de um mecanismo para a escolha automática de um balanceador de carga dada características da aplicação e da plataforma, e um mecanismo para a extração de características de aplicações paralelas para a sua emulação com COMPREHENSIVEBENCH e avaliação da possibilidade de migração da aplicação para um ambiente paralelo que permita a migração de tarefas para o balanceamento de carga.

Referências

Kale, L. V. and Krishnan, S. (1993). Charm++: A portable concurrent object oriented system based on C++. In *Proceedings of the Eighth Annual Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA 1993)*, pages 91–108. ACM.

Leung, J. Y. T. (2004). *Handbook of scheduling: algorithms, models, and performance analysis*. Chapman & Hall/CRC computer and information science series. Chapman & Hall/CRC.

Menon, H., Jain, N., Zheng, G., and Kalé, L. V. (2012). Automated Load Balancing Invocation based on Application Characteristics. In *IEEE Cluster 12*, Beijing, China.

Pilla, L. L., Ribeiro, C. P., Cordeiro, D., Mei, C., Bhatele, A., Navaux, Broquedis, F., Mehaut, J., and Kale, L. V. (2012). A Hierarchical Approach for Load Balancing on Parallel Multi-core Systems. In *Parallel Processing (ICPP), 2012 41st International Conference on*, pages 118–127.

Pilla, L. L., Ribeiro, C. P., Coucheney, P., Broquedis, F., Gaujal, B., Navaux, P. O. A., and Méaut, J.-F. (2014). A Topology-Aware Load Balancing Algorithm for Clustered Hierarchical Multi-Core Machines. *Future Generation Computer Systems*, 30(0):191–201.

Rodrigues, E. R., Navaux, P. O. A., Panetta, J., Fazenda, A., Mendes, C. L., and Kale, L. V. (2010). A Comparative Analysis of Load Balancing Algorithms Applied to a Weather Forecast Model. *Computer Architecture and High Performance Computing, Symposium on*, 0:71–78.

Tesser, R., Pilla, L. L., Navaux, P. O. A., Dupros, F., Mehaut, J. F., and Mendes, C. (2014). Using Dynamic Load Balancing to Improve the Performance of Seismic Wave Simulations. In *Parallel, Distributed and Network-Based Processing (PDP), 2014 22st Euromicro International Conference on*, pages 196–203.

Zheng, G. (2005). *Achieving high performance on extremely large parallel machines: performance prediction and load balancing*. PhD thesis, Department of Computer Science, University of Illinois at Urbana-Champaign.

Zheng, G., Bhatele, A., Meneses, E., and Kale, L. V. (2011). Periodic Hierarchical Load Balancing for Large Supercomputers. *International Journal of High Performance Computing Applications (IJHPCA)*.