Análise de desempenho da aplicação de balanceamento de carga em sistemas multiprocessadores *

Vinicius R. S. dos Santos[1], **Ana K. M. Machado**[1], **Edson L. Padoin**[1]

[1]Universidade Reg. do Noroeste do Estado do Rio G. do Sul (UNIJUI) - Ijuí - RS - Brasil

{vinicius.ribas,ana.morales,padoin}@unijui.edu.br

Resumo. *Este artigo apresenta as melhorias desenvolvidas no Balanceador de Carga AverageLB. Ele coleta informações tanto do sistema quanto da aplicação em tempo real e as utiliza nas decisões. Seu objetivo é reduzir o número de migrações de processos uma vez que migrações de processos impactam no tempo total de execução da aplicação. Os resultados apresentaram reduções significativas na taxa de migrações de objetos em comparação com outros Balanceadores de Carga.*

1. Introdução

À medida que novas simulações computacionais são desenvolvidas, aumenta a demanda por processamento dos sistemas de HPC. Um dos componentes de hardware que apresentou uma enorme evolução foi o processador, que passou a proporcionar a execução simultânea de aplicações. Assim, a programação paralela passa ser importante, possibilitando que aplicações sejam divididas em partes e executadas em paralelo nas unidades de processamento [Pilla and Meneses 2015].

A maioria das aplicações paralelas envolve comportamentos dinâmicos ou cálculos baseados em diversas fórmulas complexas. Por conta disso, empresas e instituições buscam adquirir uma infraestrutura suficiente para suportar tais aplicações [Arruda et al. 2015]. O grande problema por trás disso é que na maioria das vezes não há uma preocupação com o desbalanceamento de carga gerado por estas aplicações, impedindo que as máquinas paralelas aproveitem todo o seu potencial [Padoin et al. 2014]. Diante deste problema é que balanceadores de carga são desenvolvidos almejando um equilíbrio de cargas nos processadores.

Este trabalho busca também fazer uma análise de desempenho do AverageLB comparando o tempo total de execução e o número total de objetos migrados com mais dois Balanceadores de Carga que utilizam diferentes estratégias de tomada de decisão, disponibilizados pela plataforma do CHARM++.

O restante do trabalho está organizado da seguinte forma. A Seção 2 discute os trabalhos relacionados. A Seção 4 apresenta a metodologia utilizada na implementação e o ambiente de execução utilizado na realização dos testes. Resultados são discutidos na Seção 5, seguidos das Conclusões e Trabalhos Futuros.

2. Trabalhos Relacionados

Diversas plataformas oferecem suporte para programação com memória compartilhada e distribuída. Para a implementação do AverageLB foi utilizado o ambiente de programação paralela CHARM++. A estratégia utilizada para o balanceamento de carga é baseada na média

*Trabalho parcialmente apoiado por UNIJUI e CNPq. Pesquisa realizada no contexto do Laboratório Internacional Associado LICIA e tem recebido recursos do edital da VRPGPE de bolsa e PIBIC/UNIJUI.

aritmética dos processadores. O algoritmo busca informações sobre a quantidade de objetos mapeados em cada processador e suas cargas para calcular a carga total de cada processador [Arruda et al. 2015].

Várias aplicações científicas adotam estratégia centralizada de balanceamento de carga. Neste modelo, as decisões de balanceamento de carga são realizadas em um processador específico, com base nos dados de carga tempo de execução [Zheng et al. 2010]. Outras, por sua vez recorrem a esquemas de balanceamento de carga personalizados e constroem seu próprio modelo de carga de trabalho para orientar a atribuição de trabalho aos processos.

3. AverageLB

Alguns ambientes de programação adotam uma metodologia baseada na medição das cargas dos objetos que executam em cada processador. Para isso, o Balenceador de carga (BC) coleta automaticamente estatísticas da carga computacional e da comunicação destes objetos e armazena estas informações em um banco de dados. Este banco de dados vai ajudar o BC a decidir quando e onde migrar os objetos [Jyothi et al. 2004].

Para realizar a análise de desempenho, dentre diversos balanceadores de carga com diferentes algoritmos, o CHARM++ possui estratégias que trabalham tanto com as cargas das unidades de processamento, quanto com a comunicação entre os chares.

A estratégia para balanceamento de carga no algoritmo proposto, chamado de AverageLB, constitui-se de uma melhoria na estratégia utilizada no algoritmo GreedyLB. Essa melhoria busca equilibrar as cargas entre os processadores reduzindo o número de migrações. A implementação utiliza uma abordagem centralizada e busca atingir balanceamento levando em consideração a média aritmética das cargas de cada processador [Arruda et al. 2015].

4. Metodologia

A pesquisa proposta neste artigo tem como objetivo fazer uma análise de desempenho das melhorias desenvolvidas no Balanceador de Carga AverageLB. Para isto leva-se em consideração o tempo de execução e a quantidade de objetos migrados na execução do benchmark lb_test. O benchmark utilizado para realizar os testes foi o lb_test. Este benchmark foi escolhido por ser facilmente configurável para apresentar diferentes níveis de desbalanceamento de carga, permitindo que a carga computacional de cada tarefa seja configurada em diferentes padrões de carga tanto de irregularidade quanto de comunicação.

A plataforma em que os testes foram executados possui um processador Intel modelo i7-6500U. Cada processador possui 4 cores com 2 SMT/core, o que totaliza 8 cores. Para os testes, utilizou-se o sistema operacional Linux Manjaro 16.10 com kernel versão 4.4.33-1. A versão do CHARM++ utilizada foi 6.5.1 e do compilador g++ a versão 6.2.1.

Para analisar os resultados alcançados, os resultados dos testes foram comparados com outros dois balanceadores de carga disponíves no CHARM++, os quais são:

- **RefineLB**: que move objetos dos processadores mais sobrecarregados para os menos carregados almejando atingir uma média, sendo limitado o número do objetos migrados [Freytag et al. 2015];
- **GreedyLB**: é um algoritmo guloso, seu paradigma é frequentemente usado na teoria e na prática de otimização combinatória. Grande maioria das tarefas são migradas com

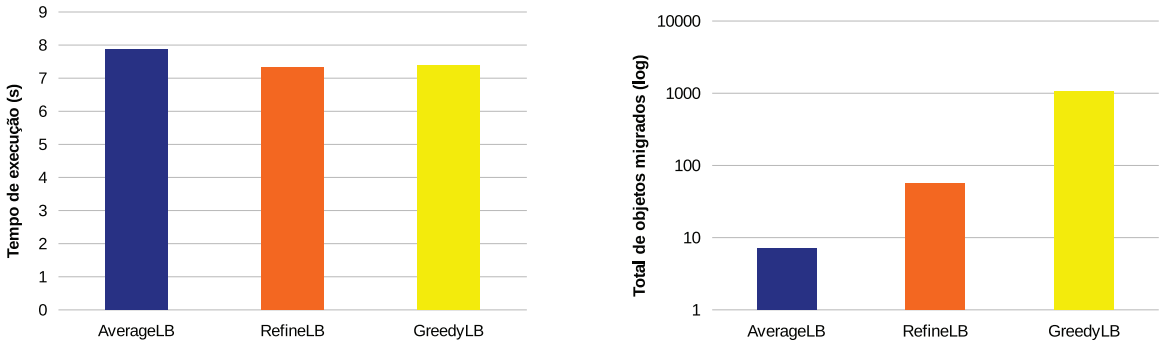
base nas informações disponíveis na iteração corrente [Bang-Jensen et al. 2004]. Em outras palavras GreedyLB busca migrar um objeto mais pesado para o processador com a menor carga até que a carga de todos os processadores esteja próxima à carga média.

5. Resultados

Para a realização dos testes, os três balanceadores foram aplicados à uma simulação utilizando 150 iterações por tarefa. As sincronizações do balanceador de carga foi definida à cada 10 iterações, sendo a quantidade de chares variada de 50 e 100 chares, com uma carga computacional que pode variar entre 500ms e 8000ms.

Durante os testes, foram levados em consideração o tempo total de execução e o total de objetos migrados entre os cores. Na Figura 1 os resultados da média das execuções com 50 chares.

Figura 1. Resultados dos testes utilizando benchmark lb_test com 50 chares.

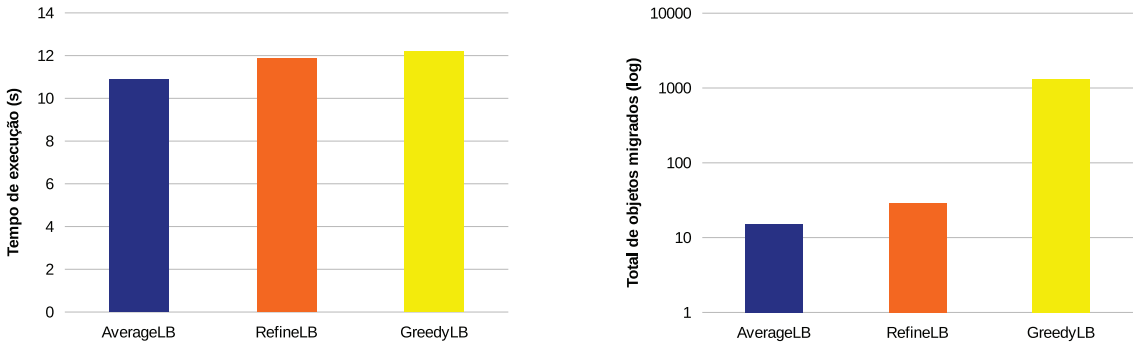


(a) Tempo total de execução (b) Total de objetos migrados

Com base nos tempos totais de execução mensurados, observa-se que com 50 chares, o menor tempo de execução foi alcançado com BC RefineLB, de 7, 34 segundos. Com o algoritmo AverageLB tem-se um tempo 7, 3% maior que o algoritmo RefineLB.

Analisando a quantidade de total de objetos migrados, percebe-se que o AverageLB apresenta a menor quantidade de migrações. Enquanto que, com GreedyLB foram migrados 1302 objetos, com AverageLB apenas 15 objetos são migrados. Esta enorme diferença ao comparado com os demais é consequência da estratégia gulosa que migra objetos do processador mais carregado para o menos carregado.

Figura 2. Resultados dos testes utilizando benchmark lb_test com 100 chares



(a) Tempo total de execução (b) Total de objetos migrados

Aumentando a quantidade de chares para 100 e analisando os resultados da média das execuções, percebe-se que o AverageLB já apresenta o menor tempo de execução. Consegue ser $9,01\%$ menor em relação ao RefineLB e $12,14\%$ menor que GreedyLB. Esta redução é alcançada principalmente em função do reduzido número de objetos migrados. Executando o benchmark com 100 chares, o AverageLB migra apenas 7 chares enquanto RefineLB e GreedyLB migraram 56 e 1077 objetos.

6. Conclusões e trabalhos futuros

Neste trabalho apresentou-se uma análise de desempenho do BC AverageLB utilizando o benchmark lb_test para a realização dos testes. Comparando os resultados com o RefineLB e o GreedyLB, concluímos que AverageLB mostrou um melhor desempenho nas migrações de objetos em ambos os testes, mostrando que o BC está realizando um bom controle de objetos migrados, evitando desperdícios e colaborando para um balanceamento de carga ágil e preciso. Também foi notado um ganho no tempo total de execução quando submetido a testes com 100 chares, apesar de ter um tempo de execução inferior quando executado os mesmos testes com 50 chares.

Como trabalhos futuros pretende-se realizar testes de desempenho utilizando maiores cargas computacionais e outros benchmarks, para uma melhor comparação dos resultados. Também objetiva-se realizar testes nas grandes máquinas paralelas, a fim de resolver problemas reais.

Referências

Arruda, G., Padoin, E. L., Pilla, L. L., Navaux, P. O. A., and Mehaut, J.-F. (2015). Proposta de balanceamento de carga para redução de migração de processos em ambientes multiprogramados. In *XVI Simpósio de Sistemas Computacionais (WSCAD-WIC)*, pages 1–8, Florianópolis, RJ.

Bang-Jensen, J., Gutin, G., and Yeo, A. (2004). When the greedy algorithm fails. *Discrete Optimization*, 1(2):121–127.

Freytag, G., Arruda, G., Martins, R. S. M., and Padoin, E. L. (2015). Análise de desempenho da paralelização do problema de caixeiro viajante. In *XV Escola Regional de Alto Desempenho (ERAD)*, pages 1–4, Gramado, RS. SBC.

Jyothi, R., Lawlor, O. S., and Kale, L. V. (2004). Debugging support for charm++. in parallel and distributed processing symposium. page 264.

Padoin, E., Castro, M., Pilla, L., Navaux, P., and Mehaut, J.-F. (2014). Saving energy by exploiting residual imbalances on iterative applications. In *High Performance Computing (HiPC), 2014 21st International Conference on*, pages 1–10.

Pilla, L. L. and Meneses, E. (2015). Programação paralela em charm++. pages 1–20.

Zheng, G., Meneses, E., Bhatele, A., and Kale, L. V. (2010). Hierarchical load balancing for charm++ applications on large supercomputers. In *2010 39th International Conference on Parallel Processing Workshops*, pages 436–444. IEEE.