Comparando diferentes ordenações de tarefas em balanceamento de carga distribuído

Vinicius Marino Calvo Torres de Freitas, Laércio Lima Pilla

Departamento de Informática e Estatística – Universidade Federal de Santa Catarina
Florianópolis – SC – Brasil

vinicius.mctf@grad.ufsc.br, laercio.pilla@ufsc.br

Resumo. *Neste artigo apresentamos um estudo sobre definição de prioridades em troca de tarefas para balanceamento de carga distribuído. O principal objetivo é entender se a aceleração na execução das estratégias, por escolher as tarefas a serem migradas mais rapidamente, compensa a menor precisão do balanceamento. Resultados mostram que, para o ambiente utilizado, o refinamento mais preciso resulta em execuções mais rápidas.*

1. Introdução

As mais custosas aplicações desenvolvidas nos meios acadêmico, científico e industrial usam de paralelismo para garantir seu desempenho. Distribuir cargas de trabalho é uma forma efetiva de escalar uma aplicação, mas isso pode trazer efeitos indesejados.

Aplicações complexas que necessitam de alto desempenho podem ter comportamento imprevisível [Pilla et al. 2014], o que pode fazer com que o primeiro escalonamento de tarefas, que as distribuiu entre diferentes núcleos computacionais, não seja eficiente ao longo de todo o tempo de execução da aplicação, causando desbalanceamento de carga. É dito que uma aplicação é desbalanceada quando uma grande quantidade de nós computacionais precisa esperar que outros terminem seu trabalho, o que causa desperdício de recursos e energia.

Uma solução para problemas de desbalanceamento de carga é o uso de algoritmos de balanceamento de carga dinâmicos, que movem tarefas entre uma sincronização e uma retomada da aplicação, buscando realocar recurso de forma a diminuir o tempo de execução total. Soluções de balanceamento de carga centralizadas e hierárquicas melhoram o desempenho de aplicações de média e baixa granularidade, porém quando se fala de larga escala (ambientes com mais de 4000 núcleos), esses algoritmos acabam tomando muito tempo para agregar toda a informação e, apesar de proverem bom balanceamento, tomam mais tempo e recursos do que é aceitável.

Algoritmos de balanceamento de carga distribuídos não são tão precisos quanto centralizados ou hierárquicos, mas trazem a vantagem de realizar menos comunicação, uma vez que fazem o balanceamento com informação local e não global. Nesse contexto, este artigo propõe um estudo sobre uma estratégia de balanceamento distribuído conhecida, o *DistributedLB* [Menon and Kalé 2013], modificando alguns aspectos do seu comportamento e avaliando seu desempenho.

2. Metodologia

O *DistributedLB* é uma estratégia distribuída em duas etapas: a primeira é uma etapa de propagação de informação, que faz uso de um *Gossip Protocol* [Demers et al. 1987]

para dissipar conhecimento sobre quais núcleos são sobrecarregados e quais são subcarregados (Figura 1); a segunda é uma etapa de troca de cargas, na qual os elementos de processamento sobrecarregados tentam enviar suas tarefas para entidades subcarregadas. Quanto maior o ambiente em questão, mais custosa será a primeira etapa, uma vez que mais informação terá de ser dissipada; já a segunda etapa tem seu peso alterado pela granularidade da aplicação, que quando cresce, faz com que mais tarefas devam ser avaliadas para migração.

No *DistributedLB* uma vez que uma tarefa é escolhida para migração, ela deve executar um protocolo semelhante ao *three-way handshake*, ou seja: um núcleo informa que quer migrar uma tarefa, um segundo decide se irá aceitá-la e envia uma mensagem de volta ao primeiro, caso a mensagem seja positiva, a tarefa é migrada, caso contrário, a transferência é considerada falha.

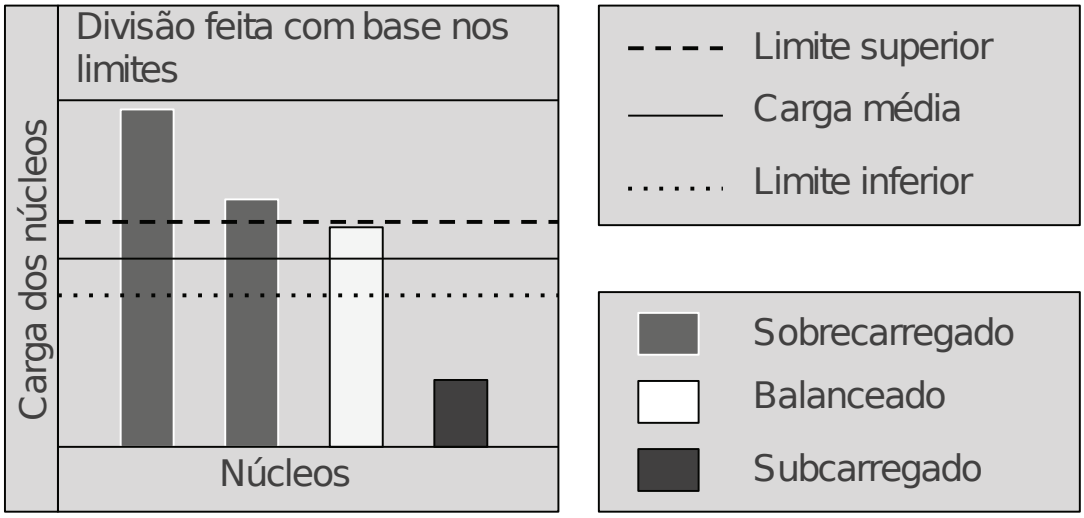

Figura 1. Desbalanceamento de carga com base em limites, divisão de núcleos entre sobrecarregados, subcarregados e balanceados com base em sua carga

Na sua implementação original, o *DistributedLB* ordenava as suas tarefas em ordem crescente de carga, fazendo as tentativas de envio nesta ordem até atingir uma carga próxima à média. Comparamos esta abordagem com mais duas.

A primeira delas é o oposto da primeira, ordenando as tarefas de maiores para menores, tendo como objetivo enviar menos tarefas. No entanto isso pode ser perigoso, uma vez que será mais difícil comportar uma tarefa grande que uma pequena. Para contornar uma situação na qual seja impossível migrar as maiores tarefas, elas não tentarão ser migradas duas vezes, sendo depois colocadas numa fila invertida, para que as tarefas menores sejam migradas numa segunda rodada. Essa implementação foi chamada de *MaxDistLB*.

A segunda delas é um ordenamento aleatório, buscando entender se uma proporção mista de tarefas pesadas e leves na sequência de tentativas gera menos falhas. A aleatoriedade pode gerar alta variância nos resultados, mas não levará a um sistema desbalanceado, uma vez que esse comportamento só se reflete na ordem das tarefas que tentará ser migrada buscando o balanceamento. Essa implementação foi chamada de *RandDistLB*.

O algoritmo original também foi reimplementado para fins de comparação, uma vez que nas novas implementações fez-se uso de bibliotecas diferentes para as estruturas de dados empregadas. A nova versão do *DistributedLB* será aqui tratada como *MinDistLB*.

3. Avaliação Experimental

A avaliação de desempenho dos algoritmos foi feita usando o ambiente paralelo CHARM++ [Acun et al. 2016]. A avaliação foi feita usando o *benchmark* sintético *lb_test* [Zheng et al. 2011] com 8192 tarefas em 150 iterações, com balanceamento de carga sendo feito a cada 30 iterações, por 40 execuções. A carga das tarefas foi variada entre 200 e 10000 ms. A comunicação das tarefas foi feita na forma de uma malha tridimensional. Estes parâmetros foram escolhidos para gerar um ambiente desbalanceado o suficiente para que o algoritmo de balanceamento pudesse trabalhar. Neste experimento, foi medido apenas o tempo de execução do perfil paralelo, sem contabilizar o tempo de *setup* do ambiente, pois este apresenta alta variância. Os experimentos utilizaram 256 núcleos (32 nós, cada um com 2 processadores *Intel Xeon E-5520 @ 2,27 GHz*)[1] rodando *Debian Jessie*. A versão do CHARM++ foi *6.7.0* e a versão do compilador GCC foi a *5.2*.

Algoritmo utilizado	Tempo médio de execução	Intervalo de confiança (+/-)
DistributedLB	$17,8587$	$0,01735$
MinDistLB	$17,8819$	$0,02035$
RandDistLB	$18,3317$	$0,02300$
MaxDistLB	$19,2308$	$0,02135$
GreedyLB	$21,3958$	$0,10594$
RefineLB	$15,5857$	$0,02392$
NullLB	$19,1687$	$0,00388$

Figura 2. Comparação dos tempos de execução (s) do *lb_test* com diferentes estratégias, ilustrando seus tempos médios e intervalos de confiança

O principal objetivo da avaliação é comparar o desempenho das três estratégias implementadas: *MinDistLB*, *MaxDistLB* e *RandDistLB*. Para fins ilustrativos, seus resultados foram comparados também a implementação original do *DistributedLB* (já que diferentes bibliotecas foram utilizadas em suas implementações), de dois algoritmos do estado da arte do balanceamento de carga centralizado (*RefineLB* e *GreedyLB*) e com uma execução sem balanceamento de carga (*NullLB*).

Os resultados apresentados na Figura 2 mostram que a estratégia *MinDistLB* se mostrou a mais rápida entre as implementadas, seguida de *RandDistLB* e *MaxDistLB*. Isso indica que, apesar da estratégia *MaxDistLB* executar menos lógica, a estratégia *MinDistLB* faz um balanceamento mais preciso por migrar tarefas mais leves. Pela mesma lógica, podemos entender que o *RandDistLB* teve um desempenho intermediário por escolher as tarefas aleatoriamente e, portanto, ter um balanceamento mais preciso que o *MaxDistLB* mas não tanto quanto o *MinDistLB*.

[1]Experimentos apresentados neste artigo foram realizados no *Grid*5000, mantido pelo grupo de propósitos científicos hospetado pela Inria e incluindo CNRS, RENATER e diversas Universidades e outras organizações (ver https://www.grid5000.fr).

Também é possível observar que o ambiente de testes não era de escala grande o suficiente para aproveitar todo o desempenho das estratégias, uma vez que não foi notado grande ganho em relação a execução sem balanceamento (*NullLB*) e *RefineLB* foi a estratégia mais eficaz (mesmo sendo centralizada), mas já se mostrou tão distribuído que *GreedyLB* não foi capaz de demonstrar um desempenho satisfatório.

4. Conclusões

Este artigo apresentou uma comparação entre diferentes prioridades de transferência de carga em balanceamento de carga distribuído, tendo como base um algoritmo do estado da arte do balanceamento distribuído. Resultados obtidos mostraram que um ajuste mais refinado em média escala gera um melhor balanceamento e um tempo total de execução menor para as aplicações.

Trabalhos futuros devem incluir testes semelhantes utilizando outros perfis paralelos e ambientes mais variados (assim como ambientes de larga escala). Este estudo pode ser usado como base para o desenvolvimento de novas estratégias distribuídas, servindo de fundamento para escolher a melhor forma de priorizar as tarefas a serem migradas.

Referências

Acun, B., Langer, A., Meneses, E., Menon, H., Sarood, O., Totoni, E., and Kalé, L. V. (2016). Power, reliability, and performance: One system to rule them all. *Computer*, 49(10):30–37.

Demers, A., Greene, D., Hauser, C., Irish, W., Larson, J., Shenker, S., Sturgis, H., Swinehart, D., and Terry, D. (1987). Epidemic algorithms for replicated database maintenance. In *Proceedings of the Sixth Annual ACM Symposium on Principles of Distributed Computing*, PODC '87, pages 1–12, New York, NY, USA. ACM.

Menon, H. and Kalé, L. (2013). A Distributed Dynamic Load Balancer for Iterative Applications. In *Proceedings of SC13: International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '13, New York, NY, USA. ACM.

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

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)*, 25(4):371–385.