Balanceamento Interno de Carga em um Ambiente Distribuído

Luiz Chaimowicz chaimo@dcc.ufmg.br
José Nagib Cotrim Árabe arabe@dcc.ufmg.br

Departamento de Ciência da Computação - ICEx - UFMG
Av. Antônio Carlos, 6627 - Pampulha - Belo Horizonte - CEP 31270-010
Telefone: (031) 499-5860 Fax: (031) 499-5858

Resumo

Este trabalho faz um estudo detalhado do sistema Dome (*Distributed Object Migration Environment*), um sistema de programação paralela para redes de *workstations*. O Dome provê uma biblioteca de objetos distribuídos que possuem mecanismos transparentes de balanceamento de carga e tolerância a falhas. Vários experimentos foram realizados em diversos ambientes de carga enfocando o mecanismo de balanceamento de carga. Para isso foi utilizado um benchmark sintético. Além dos testes com o sistema original, também foram realizadas algumas alterações no Dome com o objetivo de melhorar o seu desempenho. Os resultados obtidos mostram que o Dome é um sistema adequado para execução de programas paralelos em redes de *workstations*. O uso do seu mecanismo de balanceamento de carga com parâmetros adequados melhora bastante o desempenho dos programas, principalmente em ambientes com carga desbalanceada e instável. As modificações realizadas também produziram resultados positivos, com melhoria no desempenho do sistema na maioria dos casos.

Abstract

This work makes a detailed study of Dome (*Distributed Object Migration Environment*), a parallel programming system for a network of workstations. Dome has a library of distributed objects that have automatic load balancing and fault tolerance mechanisms. Several experiments were done in various environments with different kinds of load using a synthetic benchmark, focusing on its load balancing mechanism. Besides the tests with the original system, some modifications were made in Dome to improve its performance. The results show that Dome is a good system for parallel programming in networks of workstations. The use of the load balancing mechanism with adequate parameters greatly improves the performance, mainly in environments with unbalanced and unstable loads. The modifications also bring positive results, improving the system performance in most of the cases.

1. Introdução

Atualmente, as redes de *workstations* vêm se tornando uma alternativa cada vez mais atraente aos multiprocessadores e supercomputadores para a realização de processamento paralelo e de alto desempenho, principalmente devido ao seu baixo preço e grande disseminação. Outro fator que contribui é que as redes de comunicação estão se tornando cada vez mais rápidas. Isso, aliado ao grande avanço na capacidade de processamento das máquinas, tem melhorado muito a performance das redes de *workstations*.

Para que as redes de *workstations* possam ser usadas como computadores paralelos, são necessárias ferramentas que controlem a execução dos programas paralelos nas máquinas, preferencialmente provendo mecanismos de tolerância a falhas e balanceamento de carga. Diversas ferramentas vêm sendo construídas e estudadas em várias universidades e centros de pesquisa para a realização desse controle. Uma das ferramentas mais famosas é o PVM (*Parallel Virtual Machine*) [5], desenvolvido no *Oak Ridge National Laboratory* (ORNL). O PVM é um pacote de programas e bibliotecas que permite que uma rede de computadores heterogêneos possa ser usada como um computador paralelo virtual. O PVM possui funções para iniciar tarefas automaticamente e permitir que as tarefas se comuniquem entre si. Um projeto mais recente, que pode vir a ser o sucessor do PVM é o MPI (*Message Passing Interface*) [9]. O MPI é a tentativa de criação de uma interface padrão para o paradigma de troca de mensagens em sistemas distribuídos. Com ele, pretende-se criar um padrão que facilite o desenvolvimento e aumente a portabilidade desses sistemas. Outra ferramenta interessante é o Condor [8], desenvolvido na Universidade de Wisconsin. O Condor tem como objetivo maximizar a utilização das *workstations* aproveitando os ciclos ociosos para a execução de programas remotos, tentando interferir o menos possível com a execução dos programas locais. Basicamente, o Condor localiza *workstations* que estão ociosas e transfere para lá programas que ficam executando em *background*. Tudo isso é feito de forma transparente para os usuários. Um projeto mais abrangente é o NOW (*Network of Workstations*) [1], desenvolvido na Universidade da Califórnia, em Berkeley. Esse projeto visa a construção de um sistema para dar suporte à utilização de uma rede de *workstations* como um computador distribuído. A idéia básica é se utilizar os recursos ociosos das máquinas ligadas em rede para o processamento paralelo e de

alto desempenho. Esses recursos incluem memória, discos e não somente o processador como é comum em outros projetos.

Além dos sistemas existentes, os aspectos teóricos do balanceamento de carga externo já são bem estudados. Normalmente, ambientes multiusuário heterogêneos se tornam desbalanceados devido às diferenças na carga e na capacidade de processamento de cada máquina. Outro motivo tem origem na própria estrutura dos programas paralelos, e ocorre quando uma tarefa recebe mais trabalho do que outras. O balanceamento de carga procura distribuir o trabalho entre os processadores de forma que o tempo de execução global seja minimizado.

Um dado importante é como medir a carga de uma máquina, ou seja, como saber se uma máquina está carregada ou não. A escolha de um índice de carga adequado não é uma tarefa simples. Diversos índices de carga são estudados na literatura tais como tamanho da fila da CPU, taxa de utilização da CPU, tempo de resposta médio das tarefas, etc. Em [6], são estudados seis tipos de índices de carga. Os resultados mostram que índices simples, baseados no número de tarefas na fila da CPU, trazem bons resultados.

Praticamente todos os estudos sobre balanceamento de carga na literatura tratam do balanceamento externo. O balanceamento externo leva em conta as diferenças na carga e na capacidade de processamento das máquinas para escalonar as tarefas de modo a distribuir o trabalho de forma mais adequada. Normalmente isso é feito atribuindo-se tarefas às máquinas que estão menos carregadas. Em [7] é mostrado que existe uma grande probabilidade de que uma máquina esteja inativa enquanto tarefas são alocadas em outra que já esteja carregada. Isso mostra que distribuir o trabalho de forma adequada é de fundamental importância.

Basicamente existem dois tipos de algoritmos de balanceamento de carga externo: estáticos e dinâmicos. Nos algoritmos estáticos a atribuição das tarefas aos nodos é feito *a priori,* ou seja, as decisões sobre a atribuição de tarefas são tomadas antes do início da execução do programa. Nos algoritmos dinâmicos as decisões são tomadas durante a execução do programa, com base em informações sobre o estado atual do sistema [11]. Esses algoritmos têm um custo maior, mas esse custo é compensado com um balanceamento de carga mais eficaz, o que faz com que eles sejam mais usados. Dentre os algoritmos dinâmicos existem alguns que mudam o seu comportamento de acordo com o estado do sistema. Esses algoritmos são chamados de adaptativos. Em [4, 11] são

estudados diversos algoritmos e é mostrado que mesmo os mais simples já conseguem um grande ganho de desempenho em relação a sistemas que não utilizam o balanceamento.

Neste trabalho será estudado o Dome (*Distributed Object Migration Environment*) [2], um sistema de programação paralela para redes de workstations desenvolvido na Universidade de Carnegie Mellon. O Dome introduz um conceito novo de balanceamento de carga sobre o qual praticamente não existem trabalhos na literatura: o balanceamento de carga interno. Nele, o programa faz a redistribuição de seus dados pelas máquinas de acordo com o tempo que cada uma gastou para processá-los. O objetivo deste trabalho é fazer um estudo detalhado do sistema Dome através de testes reais, enfocando principalmente o seu mecanismo de balanceamento de carga. Os testes também procuram validar alguns dos resultados obtidos em [10], onde são feitas diversas simulações com o mecanismo de balanceamento de carga do Dome. Além disso, algumas modificações são realizadas no sistema, com o objetivo de melhorar o seu desempenho.

O restante deste trabalho está organizado da seguinte forma: a próxima seção faz uma descrição geral do sistema Dome. A seção 3 trata dos testes realizados no Dome. A seção 4 apresenta duas modificações realizadas no Dome e os testes com essa nova versões. A última seção apresenta as conclusões e as perspectivas de trabalhos futuros.

2. O Sistema Dome

O Dome é um sistema que permite ao usuário escrever programas paralelos para uma rede heterogênea de *workstations*. Ele é implementado através de uma biblioteca de classes escritas em C++ e utiliza herança e sobrecarga de operadores para tornar a programação bem simples. Em seu programa, o usuário instancia objetos dessas classes e esses objetos são automaticamente distribuídos pelas máquinas. O controle da execução e comunicação das tarefas é feito utilizando-se o PVM e é completamente transparente para o usuário. O modelo de paralelismo usado é o SPMD (*Single Program Multiple Data*), ou seja, várias instâncias do mesmo programa são criadas e cada instância executa sobre uma parte dos dados. Quando um programa escrito para o Dome vai ser executado, um ambiente de controle é criado e fica responsável pela inicialização das tarefas e distribuição inicial dos dados. O ambiente Dome criado também é

responsável pelo controle do balanceamento de carga e dos mecanismos de tolerância a falhas.

Em geral, um programa feito para o sistema Dome contém a declaração dos objetos distribuídos e uma série de operações que serão realizadas sobre esses objetos. Uma operação Dome normalmente faz com que uma função seja aplicada em paralelo a todos os elementos de um objeto distribuído. Basicamente, essas operações executam dois tipos de tarefas: processamento e comunicação. No processamento, cada máquina realiza um trabalho sobre sua parte dos dados. O tempo gasto por cada máquina é medido e é utilizado na fase de balanceamento de carga para calcular como deve ser a distribuição ideal dos elementos. Nessa fase tudo é feito localmente, não há nenhuma comunicação entre os processadores. Na fase de comunicação, os processadores trocam dados entre si para poderem obter informações que não estão disponíveis localmente.

Um dos principais atrativos do sistema Dome é o seu mecanismo de balanceamento de carga. No Dome, o balanceamento de carga é interno, ou seja, os próprios dados do programa são movimentados entre as máquinas de acordo com o tempo que cada uma gastou para processá-los. Não há migração de tarefas, apenas de dados. Portanto, o índice de carga utilizado é a taxa de execução do próprio programa Dome em cada máquina.

O funcionamento do mecanismo de balanceamento de carga no Dome é o seguinte: inicialmente, os dados são distribuídos igualmente entre os processadores não levando em conta a carga e a capacidade de processamento de cada um. Depois que um certo número de operações Dome é realizado, o programa entra na fase de balanceamento de carga. Nessa fase, é feita uma sincronização dos processos paralelos e os dados podem migrar de uma máquina para outra. As máquinas mais rápidas recebem mais dados enquanto as máquinas mais lentas passam a trabalhar com uma quantidade de dados menor. O número de operações Dome que serão realizadas entre as fases de balanceamento é um parâmetro de entrada do programa e fica fixo durante toda a execução. Como a distribuição inicial não leva em conta a carga e velocidade das máquinas, o Dome permite que a primeira fase de balanceamento possa ser feita com um intervalo menor, para evitar que o sistema fique desbalanceado por muito tempo após a distribuição inicial.

O balanceamento de carga do Dome pode ser feito de forma global ou local. No balanceamento global, todos os nodos mandam informações sobre o seu processamento

para um nodo de controle que calcula como deve ser a nova distribuição de dados e envia essa informação de volta para os outros nodos. A partir daí, os nodos vizinhos começam a trocar dados para atingir a nova configuração. A outra opção é o balanceamento local, onde as informações sobre o processamento são trocadas apenas entre os vizinhos, não havendo um nodo de controle central. A redistribuição dos dados no balanceamento global é melhor pois leva em conta informações de todo o sistema, mas seu custo é maior devido à comunicação entre os nodos. O balanceamento local pode não levar a um balanceamento perfeito após uma única fase, mas é escalável para um número maior de nodos pois não requer muita comunicação. O Dome também oferece uma outra alternativa que é realizar a primeira fase de balanceamento de forma global e as outras de forma local. Isso é interessante porque a primeira fase de balanceamento é a mais importante, pois é ela que capta o desbalanceamento existente nas máquinas no início da execução do programa.

3. Análise de Desempenho do Dome

Para realizar estudos de desempenho do sistema Dome, foi desenvolvido um *benchmark* sintético. *Benchmarks* sintéticos são programas criados artificialmente que tentam reproduzir as operações mais freqüentes (ou mais representativas) de uma grande quantidade de programas. Com eles é possível obter vários comportamentos diferentes para o sistema variando-se apenas os parâmetros de entrada do programa.

O *benchmark* sintético construído para os testes de desempenho do Dome tem dois componentes. O primeiro componente é o programa Dome que será executado. O programa é composto por operações que são freqüentemente usadas em programas típicos do Dome, mas não representa nenhuma aplicação real. Através deste *benchmark* é possível variar a quantidade de dados, o número de operações realizadas e a quantidade de comunicação requerida por essas operações. O segundo componente são geradores de carga externa com os quais é possível controlar a carga nas diversas *workstations* que farão parte do sistema. Basicamente existem dois geradores: um para carga estável e um para carga instável, que permitem que se configure o ambiente de carga desejado para os testes com o sistema Dome. Os ambientes de carga utilizados foram o balanceado estável, no qual a carga das máquinas é igual e não varia durante o tempo, o desbalanceado estável, no qual a carga é diferente nas máquinas mas ela não varia ao longo do tempo, e o ambiente instável, no qual a carga é diferente e varia ao

longo do tempo. Na maioria dos testes foram utilizadas seis máquinas, podendo ser homogêneas ou heterogêneas.

3.1 Ambiente Balanceado Estável

Durante a execução de um programa Dome em um ambiente balanceado estável, normalmente não é necessário realizar o balanceamento de carga, pois não há diferença entre o tempo de processamento das máquinas. Portanto, o primeiro experimento realizado mede o custo de se utilizar o balanceamento no Dome em uma situação onde ele não é necessário. Para isso, o tempo de execução do *benchmark* sem balanceamento de carga foi comparado com o tempo de execução utilizando-se os três tipos de balanceamento (local, global, e primeiro global depois local), variando-se os parâmetros *LBmod* e *LBinit*. *LBmod* é o número de operações Dome realizadas entre cada fase de balanceamento, e *LBinit* é o número de operações realizadas antes da primeira fase. Os resultados são mostrados no gráfico da Figura 1.

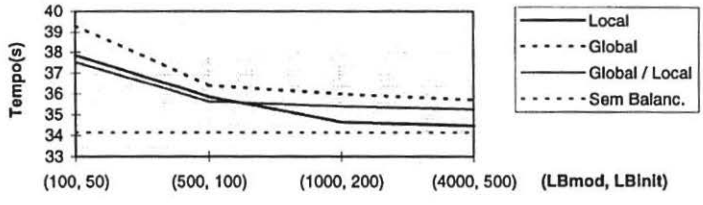

Figura 1: Balanceamento de carga – Ambiente balanceado estável

O gráfico mostra que o custo de se utilizar o balanceamento de carga em uma situação onde não é necessário é baixo: o tempo de execução ficou 15% maior para o balanceamento global (100, 50) (pior caso) e apenas 1% maior para o balanceamento local (4000, 500). A principal causa desse custo é o tempo gasto para se coletar as informações das máquinas e realizar os cálculos do balanceamento. Por isso, quanto menor o LBmod (mais fases de balanceamento) maior será o custo. Além disso, o balanceamento global tem um custo maior, pois a troca de informações é feita entre todas as máquinas, enquanto no local, a troca é feita apenas entre vizinhos. Portanto, o custo também é proporcional ao número de máquinas utilizadas. O balanceamento global/local tem um custo praticamente idêntico ao local quando LBmod é menor, se aproximando do global quando LBmod aumenta. Isso ocorre porque, com LBmod menor, o custo do primeiro balanceamento global é compensado pelas várias fases de

balanceamento local, o que não ocorre quando LBmod é alto e poucas fases de balanceamento são realizadas.

3.2 Ambiente Desbalanceado Estável

A utilização do balanceamento de carga começa a ficar vantajosa quando o ambiente de execução se torna desbalanceado. O gráfico da Figura 2 mostra os tempos de execução do Dome para um ambiente desbalanceado estável homogêneo, com os mesmos parâmetros do gráfico da Figura 1. É interessante notar que, apesar de ter um custo maior, o balanceamento global é o que tem melhor desempenho nesse ambiente. Como já foi explicado, isso ocorre porque ele leva em conta informações de todas as máquinas, conseguindo realizar um balanceamento mais eficiente. O tempo de execução com o balanceamento global (1000, 200) é 19% menor que o tempo sem balanceamento. Quando mais fases de balanceamento são realizadas (LBmod menor), o balanceamento global/local tem o melhor desempenho, pois seu custo é menor que o global e sua precisão maior que o local.

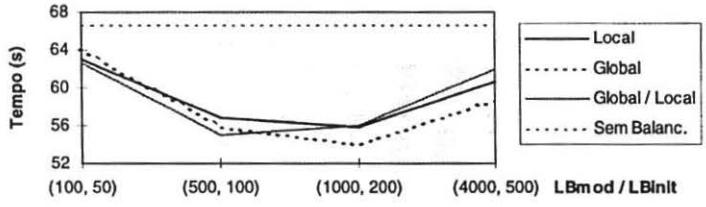

Figura 2: Balanceamento de carga – Ambiente desbalanceado estável

Um fator importante é a escolha de valores adequados para LBmod e LBinit. O valor de LBmod não pode ser muito baixo pois, como pode ser observado no gráfico da Figura 2, o custo de se realizar muitas fases de balanceamento aumenta o tempo de execução. Mas um valor muito alto pode fazer com que o sistema fique desbalanceado, principalmente com o balanceamento local onde são necessárias mais fases para conseguir o balanceamento perfeito. Para estudar melhor esses efeitos foram realizados testes variando-se apenas LBmod mantendo LBinit constante e igual a 100. Os resultados obtidos foram praticamente idênticos ao gráfico da Figura 2, onde foram variados os dois parâmetros (LBmod e LBinit). O uso do LBinit não consegue melhorar muito o desempenho do sistema pois o Dome normalmente necessita de mais fases de balanceamento para conseguir distribuir os elementos adequadamente.

O valor ideal de LBmod também varia muito em função da diferença na capacidade de processamento das máquinas. O gráfico da Figura 3 foi feito executando-se o *benchmark* com os mesmos parâmetros do gráfico anterior em um ambiente heterogêneo.

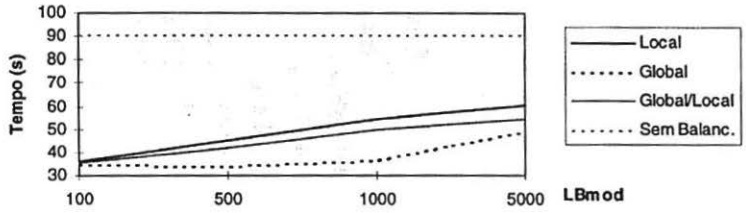

Figura 3: Variando LBmod em um ambiente desbalanceado estável heterogêneo

Em um ambiente heterogêneo a diferença no tempo de processamento das máquinas é muito maior. Enquanto o programa sem balanceamento demora aproximadamente 90 segundos para executar em uma *Sun-Sparc SLC* com carga 1, o tempo gasto em uma *IBM RS6000* sem carga é de apenas 6 segundos, ou seja a execução é 15 vezes mais rápida. No ambiente homogêneo, a diferença era apenas na carga, e o tempo de execução nas máquinas carregadas era o dobro das descarregadas. Essa diferença na capacidade de processamento das máquinas faz com que o balanceamento de carga seja de fundamental importância. O ganho obtido é de aproximadamente 63% utilizando o balanceamento global com LBmod igual a 500. Além desse fato, o gráfico da Figura 3 mostra outros aspectos interessantes. As curvas estão deslocadas para a esquerda em relação ao gráfico da Figura 2, ou seja, os tempos de execução mínimos são obtidos com valores menores de LBmod. Com a grande diferença na capacidade de processamento da máquinas, é extremamente necessário que o sistema fique balanceado o mais rápido possível. E isso é conseguido com um intervalo menor entre as fases de balanceamento. Os resultados obtidos por simulação em [10] também mostram que o valor ideal de LBmod depende muito da diferença da capacidade de processamento das máquinas.

3.3 Ambiente Instável

Ambientes com carga instável são comuns em redes de *workstations*. Isso ocorre porque normalmente as máquinas não são dedicadas exclusivamente à execução de programas paralelos, e vários usuários estão utilizando-as simultaneamente, tornando a carga instável. Como já foi explicado, para realizar os testes com o sistema Dome em

um ambiente de carga instável foi utilizado um gerador de carga artificial. Esse gerador dispara tarefas aleatoriamente nas máquinas, com demanda (D) e taxa de chegada das tarefas (λ) distribuídas exponencialmente.

Os primeiros testes foram realizados com o gerador configurado com λ igual a 30 tarefas/minuto e D igual a 5 segundos. Com essa configuração, o ambiente instável é composto por várias tarefas de duração curta pois a taxa de chegada é alta e a demanda das tarefas é baixa. Os resultados obtidos com a execução do benchmark mostram que não vale a pena realizar balanceamento de carga nesse tipo de ambiente. Com essa configuração, o ambiente torna-se muito instável fazendo com que o balanceamento de carga do Dome não funcione bem. Como a carga está variando muito freqüentemente, os cálculos que são feitos em uma fase de balanceamento de carga podem não ser válidos para a fase de execução seguinte tornando muito difícil manter o sistema balanceado. Não adianta movimentar dados entre os processadores pois a cada instante uma máquina diferente estará carregada.

Já em ambientes instáveis onde a variação da carga não é tão intensa, o balanceamento do Dome passa a valer a pena. O gráfico da Figura 4 mostra a execução do *benchmark* com os mesmos parâmetros mas em um ambiente instável com λ igual a 6 tarefas/minuto e D igual a 30 segundos, ou seja, com uma taxa de chegada menor mas com tarefas mais longas. Nesse tipo de ambiente, a instabilidade é menor. Dessa forma, o Dome consegue realizar um balanceamento de carga mais eficiente melhorando o desempenho do programa.

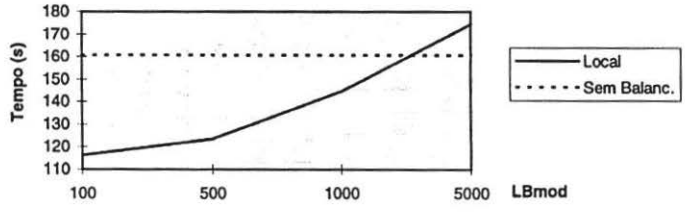

Figura 4: Balanceamento de carga – Ambiente instável (λ =6, D = 30)

O menor tempo de execução foi obtido com LBmod igual a 100, o que mostra que o intervalo entre as fases de balanceamento deve ser mais curto. Isso ocorre porque, em um ambiente instável, o sistema deve verificar freqüentemente qual a melhor distribuição de dados uma vez que a carga das máquinas está variando ao longo do tempo. Além disso, com a instabilidade, a diferença na carga das máquinas se torna

maior, o que faz com que valores mais baixos de LBmod forneçam melhores resultados, como já foi mostrado para os ambientes estáveis.

4. Modificações no Dome

4.1 Distribuição Inicial Balanceada

Quando os objetos distribuídos são criados, o Dome divide seus elementos igualmente entre os processadores, não levando em conta se existe diferença na carga ou na velocidade de processamento das máquinas. Isso não é interessante em ambientes desbalanceados, pois a execução do programa fica prejudicada devido ao desbalanceamento inicial de seus dados. A única alternativa que o Dome oferece para contornar essa situação é fazer a primeira fase de balanceamento de carga após a realização de número menor de operações, de acordo com o parâmetro LBinit. Mas o Dome necessita de mais de uma fase de balanceamento de carga para atingir o balanceamento perfeito o que faz com que a utilização de LBinit não seja suficiente para acabar rapidamente com o desbalanceamento inicial das máquinas.

Uma alternativa para solucionar esse problema é realizar a distribuição inicial dos dados de forma balanceada (DIB), ou seja, atribuir às máquinas quantidades diferentes de elementos de acordo com a capacidade de processamento de cada uma. Para que isso seja feito, é necessária a utilização de um índice de carga que mostre a capacidade de processamento das máquinas no início da execução do programa, indicando qual é a melhor forma de distribuir os dados. Com isso, espera-se diminuir o desbalanceamento inicial na distribuição de dados entre os processadores.

Para realizar a distribuição inicial de forma balanceada, ao invés dos dados serem distribuídos igualmente entre os processadores, eles passaram a ser distribuídos de acordo com uma função que indica qual é o custo de se realizar o processamento dos dados em cada máquina. Quando o ambiente Dome é inicializado, cada processador calcula o seu valor para a função de custo e o envia para um processador mestre que armazena esses valores. No momento que um vetor distribuído é criado, o processador mestre calcula o número de elementos que cada processador irá receber de acordo com os valores da função de custo que ele tem armazenado. Duas funções de custo foram estudadas. As duas são semelhantes e se baseiam principalmente no tamanho da fila e na velocidade das máquinas. A função que foi escolhida é a razão entre o número médio

de tarefas na fila durante o último minuto (fornecida pelo comando *uptime* do Unix) e a velocidade relativa do processador em relação a um processador padrão [3].

Depois de escolhida a função de custo, foram realizados diversos testes para estudar se a utilização da distribuição inicial balanceada traz alguma melhoria no desempenho do sistema Dome. O primeiro teste foi feito em um ambiente desbalanceado estável com 6 máquinas homogêneas, utilizando-se balanceamento global, local e sem balanceamento de carga. Os resultados podem ser observados no gráfico da Figura 5.

O gráfico traz uma série de resultados interessantes. O primeiro deles, e mais significativo, é que a distribuição inicial balanceada melhorou o desempenho do Dome em todos os casos. Na execução sem balanceamento de carga o ganho foi de 25%, ficando em 7% para o balanceamento local e 4% para o balanceamento global. O uso da DIB faz com que o sistema trabalhe de forma balanceada desde o princípio e isso melhora o desempenho, pois não é mais necessário esperar algumas fases de balanceamento para que o sistema atinja um estado ótimo. Outro resultado interessante é que o tempo de execução com a DIB foi praticamente igual, tanto na execução sem balanceamento de carga como também quando foram utilizados os balanceamentos local e global. Isso mostra que em um ambiente desbalanceado estável, a distribuição inicial balanceada já é suficiente para eliminar as diferenças que existem entre as máquinas, não sendo mais tão necessária a realização de balanceamento de carga.

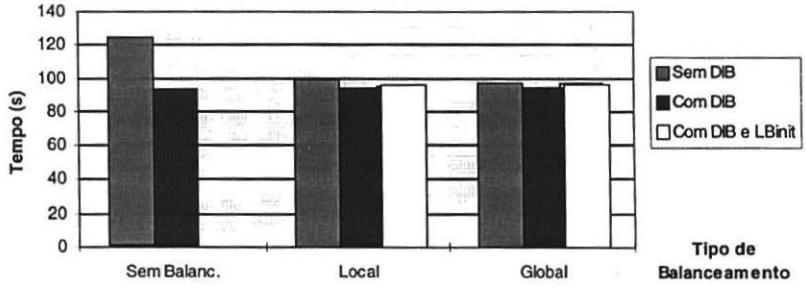

Figura 5: DIB em um ambiente desbalanceado estável homogêneo

É importante lembrar que a distribuição inicial balanceada possui um certo custo que não foi mostrado no gráfico anterior. Para realizar os cálculos da DIB, é necessário que cada máquina calcule o seu valor para a função de custo e o envie para um processador mestre, que determinará quantos elementos cada máquina deve receber. Esse processo é feito durante a inicialização do ambiente Dome e aumenta um pouco o

custo dessa fase. Mas o custo de inicialização do Dome depende apenas do número de máquinas, e é muito pequeno comparando-se ao tempo total de processamento [3].

4.2 LBmod Variável

O balanceamento de carga do Dome é muito sensível ao tipo de ambiente em que o programa está executando. Foi mostrado na seção 3 que o tempo de execução dos programas atinge o seu valor mínimo para diferentes valores de LBmod, de acordo com a variação da carga e com a diferença de velocidade das máquinas que compõem o sistema. De modo geral, quanto maior a diferença entre as máquinas, menor deve ser o intervalo entre as fases de balanceamento. Ambientes instáveis também requerem um valor menor para LBmod, a não ser que a instabilidade seja muito alta, quando não vale a pena fazer o balanceamento de carga. Em ambientes desbalanceados estáveis homogêneos, o valor de LBmod deve ser mais alto assim como em ambientes sem carga, onde não é necessário o uso de balanceamento. Além disso, em ambientes estáveis o valor ótimo de LBmod também depende do número de total de operações que o programa irá executar. Não importa apenas o número de operações entre as fases de balanceamento mas também o número de fases que serão executadas. A realização de muitas fases de balanceamento em um ambiente estável pode levar a uma perda de desempenho, pois normalmente o sistema já se torna balanceado a partir de certo ponto da execução.

Essa grande variação torna a escolha de um valor adequado para o LBmod uma tarefa difícil. Uma solução para isso é fazer com que o valor de LBmod seja variável, se adaptando automaticamente à heterogeneidade e instabilidade dos ambientes onde o programa Dome esteja executando. Com isso, espera-se obter uma melhoria no desempenho, pois o valor de LBmod se aproximará do ideal para qualquer tipo de ambiente.

Existem basicamente duas dificuldades para a implementação do LBmod variável. A primeira delas é como medir a instabilidade e a heterogeneidade do ambiente e a segunda, e mais complexa, é como variar o LBmod em função da instabilidade medida. Para medir a instabilidade e heterogeneidade do ambiente será calculado um coeficiente baseado na diferença entre os tempos de processamento das máquinas. O coeficiente será obtido pela diferença proporcional entre o tempo gasto pela máquina mais lenta e o tempo gasto pela máquina mais rápida: $c = (T_{max} - T_{min}) / T_{min}$.. Esse coeficiente indica

exatamente quantas vezes uma máquina é mais lenta que a outra. Portanto, quanto maior for a diferença entre as máquinas, maior será o coeficiente.

De posse desse coeficiente, é necessário definir uma função que o relacione com a variação do LBmod. A princípio, quanto maior o coeficiente menor deve ser o valor de LBmod e vice-versa Mas é muito difícil obter uma função que relacione esses dois valores de forma exata. Além disso, variações pequenas no valor de LBmod não interferem muito no desempenho geral do sistema. Por isso, ao invés de uma função, foi utilizada uma máquina de estados finita para representar a variação do LBmod de acordo com o coeficiente. LBmod poderá assumir cinco valores diferentes: 100, 500, 1000, 5000 e 10000 dependendo do valor do coeficiente *c*. De acordo com a variação de *c*, o valor de LBmod passará de um estado para outro da máquina, como pode ser visto na Figura 6.

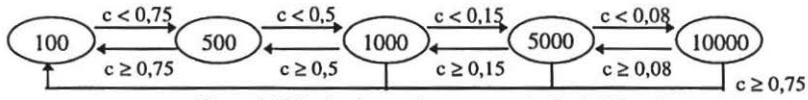

Figura 6: Máquina de estados para a variação de LBmod.

O sistema sempre iniciará sua execução no estado onde LBmod é igual a 100. A partir daí, o aumento será sempre gradual, ou seja, para atingir o seu último estado a máquina terá que passar por todos os estados intermediários. Com isso, tenta-se garantir que o sistema esteja realmente balanceado ao atingir o estado onde LBmod é igual a 10000. Quando a instabilidade aumenta, a máquina percorre o sentido inverso passando de valores maiores para valores menores de LBmod. Mas independentemente do estado no qual a máquina estiver, um aumento muito grande da instabilidade faz com que o valor de LBmod seja colocado novamente em 100, para garantir que essa instabilidade repentina seja tratada rapidamente.

Para verificar o funcionamento do LBmod variável foram feitos testes em todos os ambientes de carga: balanceado estável, desbalanceado estável com máquinas homogêneas e heterogêneas e ambiente instável ($\lambda = 6$, $D = 30$). A execução com o LBmod variável foi comparada com a execução utilizando-se LBmod igual a 1000, LBmod igual a 100 e também com a execução sem balanceamento de carga. O número de iterações do *benchmark* foi maior do que nos experimentos anteriores para poder se observar melhor os efeitos da variação do LBmod. Os resultados dos experimentos podem ser observados no gráfico da Figura 7.

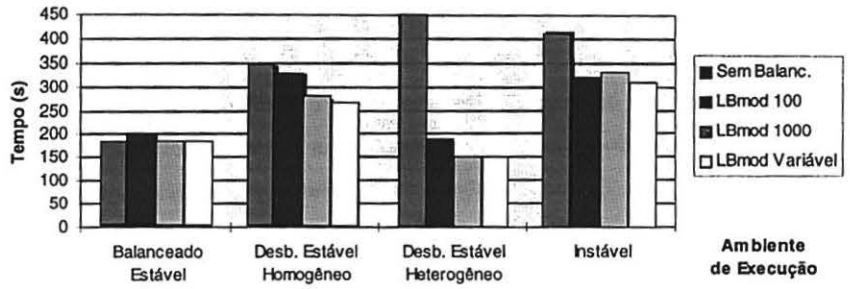

Figura 7: Desempenho do LBmod variável

O gráfico traz resultados bem interessantes. No ambiente balanceado estável, o melhor desempenho foi obtido sem o uso do balanceamento de carga. Logo em seguida vêm o LBmod variável e o LBmod 1000 com uma diferença de menos de 0,5% entre eles. Nos ambientes desbalanceados estáveis homogêneo e heterogêneo, o melhor desempenho foi obtido pelo LBmod variável seguido pelo LBmod 1000. Por fim, no ambiente instável, o melhor desempenho também foi obtido pelo LBmod variável seguido pela execução com LBmod igual a 100.

Portanto, a execução com o LBmod variável teve o melhor desempenho em todas as situações onde o balanceamento de carga é necessário. Além disso, ele também foi o melhor, dentre os balanceamentos, no ambiente estável. A variação do LBmod faz com que o mecanismo de balanceamento de carga se adapte às variações de carga. Quando a carga estiver instável, o valor do LBmod se torna menor, procurando manter o sistema sempre balanceado. Quando a carga se torna estável, menos fases de balanceamento são realizadas (LBmod maior) diminuindo-se o tempo total de execução. Ou seja, o valor de LBmod estará sempre próximo do valor ideal para qualquer tipo de ambiente.

Esses resultados indicam que o uso do LBmod variável no balanceamento de carga do Dome é importante. Além de trazer ganhos de desempenho em todos os ambientes de carga, o LBmod variável faz com que o mecanismo de balanceamento de carga se torne ainda mais transparente para o usuário. Ele não terá mais que se preocupar em configurar os parâmetros do balanceamento pois isso será feito automaticamente de acordo com o ambiente de carga no qual o programa estiver executando.

5. Conclusões e Perspectivas Futuras

Esse trabalho realizou um estudo detalhado do sistema Dome, enfocando principalmente o seu mecanismo de balanceamento de carga. Foram realizados

experimentos com um benchmark sintético em diversos ambientes de carga e com base nos resultados foram propostas mudanças ao sistema. Os resultados indicaram que o balanceamento interno de carga no Dome melhora o desempenho dos programas na maioria dos casos, sendo de fundamental importância em ambientes desbalanceados e em ambientes instáveis. As modificações realizadas no Dome também trouxeram bons resultados, indicando que alterações simples podem melhorar o desempenho das aplicações paralelas.

Este trabalho deixa algumas perspectivas futuras. Diversos testes ainda podem ser feitos com o Dome, principalmente com um número maior de máquinas, e utilizando-se programas reais. Além disso, melhoramentos podem ser feitos nas modificações realizadas no Dome, como por exemplo, o estudo de outros índices de carga para a distribuição inicial balanceada, e a utilização de outras técnicas para a variação do LBmod.

Referências

[1] Anderson, T., Culler, D., Patterson, D. and the NOW Team, "A Case for NOW (Networks of Workstations)", *IEEE Micro*, February 1995, pp.54–64.

[2] Árabe, J. N. C., Beguelin, A., Lowekamp, B., Seligman, E., Starkey, M. and Stephan, P., "Dome: Parallel Programming in a Distributed Computer Environment", *Proceedings of 10th the International Parallel Processing Symposium*, Honolulu, Hawaii, April 15–19, 1996, pp. 218–224.

[3] Chaimowicz, L., "Balanceamento Interno de Carga em Redes de Workstations: Um estudo do Sistema Dome", *Dissertação de Mestrado do Departamento de Ciência da Computação da UFMG*, Setembro de 1996.

[4] Eager, D. L., Lazowska, E. D. and Zahorjan, J., "Adaptive Load Sharing in Homogeneous Distributed Systems", *IEEE Transactions on Software Engineering*, Vol. SE–12, No. 5, May 1986, pp. 662–675.

[5] Geist, A., Beguelin, A., Dongarra, J., Jiang, W., Manchek, R., and Sunderam, V., "*PVM: Parallel Virtual Machine – A User's Guide and Tutorial for Networked Parallel Computing*", The MIT Press, Massachusetts, 1994.

[6] Kunz, T., "The Influence of Different Workload Descriptions on a Heuristic Load Balancing Scheme", *IEEE Transactions on Software Engineering*, Vol. SE-17, No. 7, July 1991, pp.725–730.

[7] Livny, M. and Melman, M., "Load Balancing in Homogeneous Broadcast Distributed Systems", *Proceedings of the ACM Computer Network Performance Symposium*, 1982, pp. 47–55.

[8] Litzkow, M. J., Livny, M. and Mutka, M. W., "Condor – A Hunter of Idle Workstations", *Proceedings of the 8th International Conference on Distributed Computing Systems*, IEEE Computer Society Press, Los Alamitos, 1988, pp.104–111.

[9] Message Passing Interface Forum, "MPI: A Message-Passing Interface Standard", Technical Report CS–94–230, University of Tennessee, Knoxville, TN, April 1994.

[10] Murta, C. D. e Árabe, J. N. C., "Auto Balanceamento de Carga em Programas Paralelos", *Anais do VIII Simpósio Brasileiro de Arquitetura de Computadores e Processamento de Alto Desempenho*, Recife, Agosto de 1996, pp. 161–171.

[11] Shivaratri, N. G., Krueger, P. and Singhal, M., "Load Distributing for Locally Distributed Systems", *IEEE Computer*, Vol. 25, No. 12, December 1992, pp. 33–44.