Balanceamento de Carga em *Clusters* por Replicação de Dados Sob Demanda em um SGBDD

REINALDO LOURENSO SERGIO TAKEO KOFUJI
Laboratório de Sistemas Integráveis – Escola Politécnica
Universidade de São Paulo
São Paulo - SP- Brasil
CEP 05508-900
(reinaldo, kofuji)@lsi.usp.br

ABSTRACT

In the last years, a significant growth of network traffic and processing load has been verified, mainly in database servers. Many factors have been responsible for this growth, such as: growth on demand for information availability, and increase of both size and quantity of stored information, as can be verified in multimedia application.

One solution for the problem of processing large volume of information is the utilization of cluster based servers, working together with distributed database systems. This type of solution allows the parallel manipulation of great quantities of information.

Although multiprocessing in clusters systems is, in fact, being used more and more due to high performance, low cost and high scalability characteristics, we verified that specific database systems for these architectures present some restrictions in relation to data replication, and load balance.

The main objective of this work is the study and implementation of a mechanism that allows a better cluster load balance, in periods of high demand for information, by replicating the most requested data in a Distributed Database Management System (DDBMS).

The complete process of copies generation is autonomous, with dynamic distribution of copies for the idle or less loaded node.

1. INTRODUÇÃO

Nos últimos anos, tem-se verificado um crescimento significativo do tráfego em redes e da carga de processamento, principalmente nos servidores de banco de dados. Muitos fatores têm sido apontados como responsáveis por este crescimento, destacando-se: (*i*) Demanda cada vez maior por disponibilidade de informações; (*ii*) Aumento tanto do tamanho quanto da quantidade de informações armazenadas; (*iii*) Consultas cada vez mais complexas; (*iv*) Diversificação dos tipos de informações armazenadas, como texto, imagem, áudio e vídeo.

Sistemas multiprocessados baseados em *clusters* de servidores, trabalhando em conjunto com sistemas de banco de dados distribuídos, foram apresentados como uma tendência de solução para o problema de processamento de grandes volumes de informações. A união destes dois sistemas permite a manipulação, em paralelo, de grandes volumes de informações [1].

Outra solução que permite o aumento da disponibilidade de dados e tolerância a falhas é o emprego de replicação de dados. Muitas pesquisas têm sido realizadas sobre gerenciamento de dados replicados nos mais diversos sistemas de banco de dados ou sistemas de arquivos distribuídos.

Contudo, essas soluções apresentam problemas. Foi verificado que sistemas de banco de dados específicos para arquiteturas de clusters apresentam algumas limitações em relação à replicação de dados [5] e balanceamento de carga [6] [7] [8] [11]. Um dos motivos para isto, é que estes sistemas consideram o *cluster* como uma máquina única e, portanto, não disponibilizando ou restringindo a replicação de dados. Em relação ao balanceamento de carga, SGBD para *clusters*, adotam esquemas de balanceamento de carga estáticos, onde o balanceamento está fortemente ligado à forma de distribuição dos dados no *cluster*.

Outro grave problema é a consistência das réplicas, espalhadas por um grande número de servidores, quando se efetua atualizações [2].

Este cenário é complexo e mostra que propostas tradicionais são inviáveis para certos tipos de aplicações não convencionais. Contudo, estas aplicações estão se tornando cada vez mais comuns, como é o caso das aplicações multimídia que estão emergindo na educação, disseminação de informações, entretenimento e muitas outras aplicações [9] [10].

Uma das possíveis soluções, para tentar minimizar esse quadro, são os modelos de replicação autônoma com disseminação dinâmica das cópias, que vem sendo propostos com a finalidade de melhorar o gerenciamento de recursos disponíveis em

sistemas distribuídos. Replicação autônoma também fornece uma solução interessante para a disponibilidade de dados, a consistência de dados replicados e o balanceamento de carga entre os servidores. [9] [10] [3] [4].

James S.Gwertzman [3] propõe, no seu trabalho, a replicação de dados sob demanda, onde somente os dados mais requisitados são replicados em caches de um sistema de arquivos distribuídos. O mesmo conceito foi utilizado neste trabalho, contudo, em um SGBDD.

ChengFu Chou [10] apresenta um modelo de replicação de dados dinâmico para fornecer balanceamento de carga e um melhor gerenciamento dos recursos disponíveis em um servidor de mídia. As réplicas de dados são criadas quando o servidor não estiver disponível. Alguns dos cálculos utilizados por ChengFu Chou, foram aproveitados neste trabalho, com algumas mudanças. Entretanto, para o nosso trabalho, o documento mais requisitado foi considerado.

Em um outro trabalho, Ouri Wolfson [4] apresenta um algoritmo para replicação dinâmica de dados a fim de propiciar um melhor desempenho de esquemas de replicação em SGBDD. Esse algoritmo se adapta mudando o esquema de replicação de objetos dinamicamente. No nosso trabalho, o algoritmo se adapta considerando a disponibilidade de cada servidor, ou seja, o nível de carga para onde os documentos serão replicados.

2. PROPOSTA DESTE TRABALHO

Este trabalho uniu três propostas: despachante central[11], replicação de dados sob demanda autônoma[3] e disseminação dinâmica de réplicas[5] em um único mecanismo. Esta união visou propiciar uma geração automática de cópias de dados, sob demanda, em um SGBDD instalado em um *cluster* e com isso, efetuar o balanceamento de carga durante os períodos de maior demanda por uma determinada informação.

O modelo de despachante central foi utilizado na seleção do servidor por intermédio de uma tabela de *meta-dados* localizada no despachante. Esta tabela contém o endereço de cada documento existente no sistema, bem como o valor de carga de cada servidor. Neste método de escolha, atende à requisição o servidor que tiver a menor carga e que possuir o documento desejado. Esta tabela também é utilizada para selecionar qual servidor receberá a réplica do dado.

Como o despachante pode ser uma fonte de problemas, devido a este ser único e portanto um possível gargalo, a função de gerenciamento do mecanismo é distribuída entre os participantes, permitindo uma certa autonomia destes, e aliviando a carga do despachante, que funciona como um roteador inteligente. O despachante pode ainda ser espelhado, a fim de prever falhas ocasionais.

A replicação de dados sob demanda autônoma[3], baseia-se na replicação do dado mais requisitado, dentro de um certo intervalo de tempo definido de acordo com a carga em cada servidor, gerada pelo excesso de requisições. Considerou-se, também, o método de disseminação dinâmica das réplicas para os servidores de menor carga [5].

A proposta de replicação de dados sob demanda em um SGBDD baseado em *clusters*, além de proporcionar balanceamento de carga, independentemente do esquema de distribuição de dados utilizado, durante os períodos de maior demanda por informação, pode otimizar ainda mais os recursos disponíveis se, para aqueles dados cuja demanda decrescer, suas cópias forem eliminadas.

Isso, além de liberar espaço em disco, minimiza e restringe bastante o problema de consistência dos dados àqueles cuja demanda se mantém elevada.

A figura 1 apresenta o esquema de funcionamento do mecanismo, onde inicialmente é feita uma requisição para descobrir que servidor irá atender à requisição. Em seguida, após obter o endereço do servidor selecionado, o cliente requisita diretamente a este servidor o documento desejado.

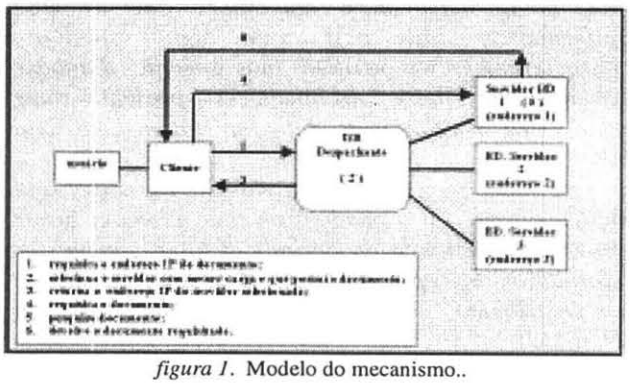

figura 1. Modelo do mecanismo..

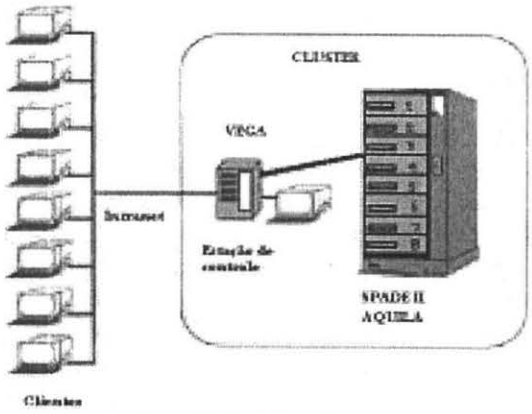

figura 2. Ambiente de teste.

3. DESCRIÇÃO DO CENÁRIO

Para o cenário utilizado neste trabalho figura 2, foi adotado como equipamento base um sistema de *cluster* de nome SPADE-II [12], onde SPADE é acrônimo de Sistema de Processamento de Alto Desempenho, designando uma família de sistemas de processamento paralelo de larga escala baseada em estações de trabalho e interconexão ponto-a-ponto de alta velocidade, com suporte à memória compartilhada e à passagem de mensagens. O SPADE-II é uma arquitetura genérica composta basicamente de estações de trabalho multiprocessadoras, interligadas através de um sistema de interconexão hierárquico baseado em comutadores e enlaces Myrinet.

A configuração pode variar de, no mínimo, "três" a, no máximo, "oito" nós por *rack*, com um dos nós sendo o de controle (despachante), como mostra a figura 2. O número de *racks* também pode ser variável, dependendo da utilização.

A configuração escolhida para este trabalho usou apenas três dos oitos nós de um único *cluster*, além do nó de controle, em virtude de limitações no ambiente de testes, como por exemplo, dificuldade de alocação de mais nós exclusivos ao presente trabalho devido a outros trabalhos em andamento.

Cada nó é um processador QuadPentiumPró de 200 MHz, equipado com 6 GB de HD IDE, 256KB de Cache e 1GB de RAM. A utilização de nós multiprocessados SMP, deve-se ao SGBD adotado, conseguir utilizar melhor essa arquitetura em virtude de sua capacidade de execução com múltiplos threads.

O *cluster* possui três redes de interconexão: uma rede primária para controle, Fast-Ethernet 100BaseT, uma secundária para dados, Myrinet ou SCI e uma terciária (opcional) para sincronização [12] O sistema operacional utilizado foi o *LINUX REDHAT* 6.0 (*Kernel*- 2.2.5).

O nó de controle é equipado com um *Quad-Pentium-Pro(AMIGoliath)*,1 GB RAM, 8 GB HD SCSI, uma unidade de CD-ROM, um *Jaz Drive*, 20' Color Monitor, uma unidade de fita DAT 8 GB.

4. DESCRIÇÃO DO MECANISMO

O mecanismo foi implementado utilizando-se o SGBD *Informix-DS LINUX Edition 7.30.UC5*, por este permitir operações em ambiente distribuído, replicação de dados, processamento paralelo dentro de um nó SMP, bem como armazenamento de grandes objetos binários (*BLOBS - Binary Large Objects)*, com tamanhos de até 2 gigabytes cada um.

É importante ressaltar que até o início deste trabalho, outros SGBD disponíveis para *Linux* ainda não apresentavam os recursos descritos para o *Informix-DS*.

Para o estudo do comportamento do mecanismo implementado, durante a realização dos testes, a aplicação utilizada foi um servidor de imagens de alta definição (imagens estáticas *jpeg*). Contudo, o sistema pode ser utilizado para outros tipos de implementação como servidor de *stream* vídeo, servidor de áudio ou servidor *WEB*.

O mecanismo é constituído por: tabelas de controle, tabelas de dados, agentes de monitoramento, processo de controle de acesso, algoritmo para pesquisa dos documentos requisitados, processo de cálculo do *limite* de replicação, processo de replicação e processos de geração de tráfego.

O protótipo é controlado via parâmetros, os quais podem ser configurados de acordo com as necessidades e características do sistema em que ele esteja operando. Foram considerados como parâmetros os limites impostos pelo sistema tais como: (*i*) taxa máxima de transferência do disco; (*ii*) taxa máxima de transferência da rede; (*iii*) quantidade máxima de acessos concorrentes suportada pelo sistema de banco de dados.

Os agentes de monitoramento são os principais elementos do sistema. Sua função é medir continuamente a carga em cada servidor de banco de dados do *cluster*, bem como disparar os processos de cálculo do limite de replicação quando algum servidor atingir um determinado valor de carga. Se a carga se alterar entre duas medidas, ela será atualizada na tabela de endereços no nó de controle (despachante).

5. FUNCIONAMENTO DO MECANISMO

Como todas as requisições têm que passar, obrigatoriamente, pelo nó de controle, portanto, este assume todo o processamento inicial.

Com todos os nós servidores ativos, os agentes de monitoramento, conforme mencionado anteriormente, acompanham continuamente a carga em cada um dos servidores do *cluster*. Esse monitoramento contínuo fornece os dados estatísticos necessários para que os processos de pesquisa efetuem com mais eficiência uma melhor escolha do servidor que poderá atender às requisições recebidas.

Com a chegada de uma requisição, é realizada uma pesquisa na tabela de índice mestre em conjunto com a tabela de endereços para obter o servidor com a menor carga e que possui o documento requisitado. Em seguida, é disparado, em paralelo, dois processos: um para registrar a requisição feita na tabela de acessos; e outro, para obter o documento no servidor

selecionado. Ambos os processos são criados no lado do cliente.

Tentou-se distribuir o processamento para os nós servidores, ficando para o nó de controle somente a tarefa de localizar o documento e rotear as requisições. Portanto, o processo de controle de acessos é direcionado para o mesmo servidor em que o processo de pesquisa do documento foi iniciado. O processo de verificação do documento mais requisitado também é executado em cada servidor, separadamente.

Quando o volume de requisições for muito elevado para um determinado servidor, isso se refletirá na sua carga. Caso esta carga ultrapasse um valor pré-determinado, será disparado o processo para verificação do documento mais requisitado; se o número de requisições encontrado dentro de um certo intervalo de tempo ultrapassar o valor do *threshold* pré-calculado, será disparado o processo de replicação no nó de menor carga do *cluster*.

O tráfego de requisições foi gerado aleatoriamente por até três estações cliente ao mesmo tempo, que estavam localizadas fora do *cluster*, na *intranet*, como mostra a figura 2.

A consistência das cópias criadas foi considerada. Contudo, não foram implementadas, nesta fase do projeto, nem a atualização e nem a eliminação dessas cópias.

6. TESTES E RESULTADOS

O mecanismo proposto neste trabalho permitiu efetuar a distribuição das requisições feitas para um determinado documento, muito requisitado, entre os servidores, promovendo com isso balanceamento de carga no *cluster*, durante os períodos de maior demanda por informação. O balanceamento de carga foi possível, pois o mecanismo efetuou com sucesso: (*i*) Avaliação da carga em cada um dos servidores; (*ii*) Determinação do excesso de carga nos servidores; (*iii*) Determinação e replicação para um servidor com menos carga (ou ocioso), do documento mais requisitado.

O processo de teste inicia-se com a chegada de uma requisição ao despachante, conforme descrito no item anterior.

Procurou-se gerar o tráfego de requisições sempre no limite de acessos concorrentes permitido pelo banco de dados.

A geração de tráfego foi feita por processos que efetuam requisições a um ou mais documentos em paralelo. Tanto o documento, como o número de iterações que os processos realizam, eram informados no início do processamento.

Foram criados três processos de geração de requisições, cada um deles possuindo um método diferente para gerar requisições.

Um dos métodos gera requisições seqüenciais para um determinado documento em um nó específico. O tráfego é seqüencial porque uma requisição só é iniciada após a requisição anterior ter sido encerrada; este método permite determinar o tempo total necessário para transferir um documento ou "*n*" documentos seqüencialmente. Este processo registra, em um arquivo de saída, o documento escolhido, o número de iterações especificadas e o tempo total gasto para todas as iterações.

Ressalta-se que o tempo total gasto deve ser entendido como sendo o tempo necessário para conectar-se ao banco de dados de controle, efetuar o acesso ao índice mestre, conectar-se ao banco de dados do documento, e efetuar o acesso à tabela de documentos e transferir o documento para a estação cliente.

O segundo método gera requisições em paralelo para três documentos distintos, estando eles em um único nó ou em nós distintos. As requisições são geradas sempre de três em três (por este ser o número total de nós utilizados neste trabalho). Contudo, entre estes disparos, o processo aguarda um intervalo de tempo pré-determinado. Este processo permite que se mantenha todo o sistema com carga durante um intervalo de tempo maior.

O terceiro método gera requisições em paralelo para um documento específico, no servidor com menor carga que possuir este documento. Este processo utiliza o algoritmo de pesquisa descrito anteriormente. Com ele, pode-se verificar o balanceamento de carga entre os nós do *cluster*.

Os parâmetros de tempo, para escolha do intervalo a ser selecionado na tabela de acessos (log.), foram especificados para extrair os acessos em intervalos de tempo pequenos de no máximo 1 (um) minuto. Por este motivo, conseguiu-se obter resultados com um número reduzido de requisições, em média entre 60 a 90, não necessitando, portanto, de um número muito elevado de requisições para cada teste.

Os gráficos apresentam os resultados obtidos de carga em função do tempo para o teste que provocou excesso de requisições para um documento localizado no servidor 1. Na figura 3, pode-se observar a carga total gerada por servidor do *cluster;* essa carga foi gerada aleatoriamente para diversos documentos do cluster, mas um número maior de requisições foram geradas para um determinado documento.

A figura 4 apresenta a carga total gerada pelo documento mais requisitado no *cluster*, sem especificar qual dos servidores atendeu a essas requisições. Inicialmente, como já mencionado, o

documento mais requisitado encontrava-se somente no servidor 1.

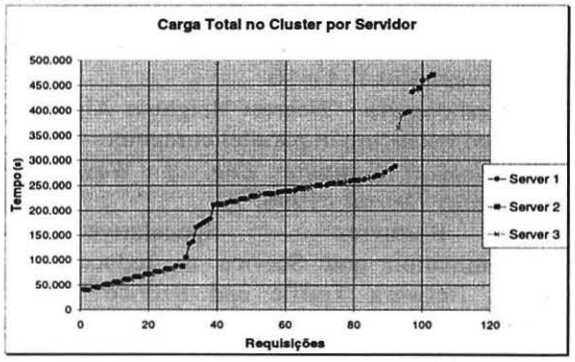

figura 3. Carga total no cluster por servidor.

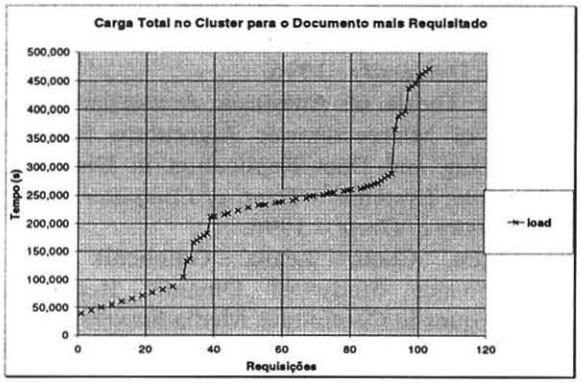

figura 4. Carga total no cluster para o documento mais requisitado.

Na figura 5, pode-se observar a carga total gerada para o documento mais requisitado, separada por servidor, onde em um determinado instante nota-se que as requisições para o documento em questão começam a ser distribuídas entre os demais servidores.

Por final, na figura 6 é apresentada somente a carga adicional gerada no *cluster*, e que foi considerada pelo mecanismo quando da replicação do documento mais requisitado e também para a distribuição das requisições.

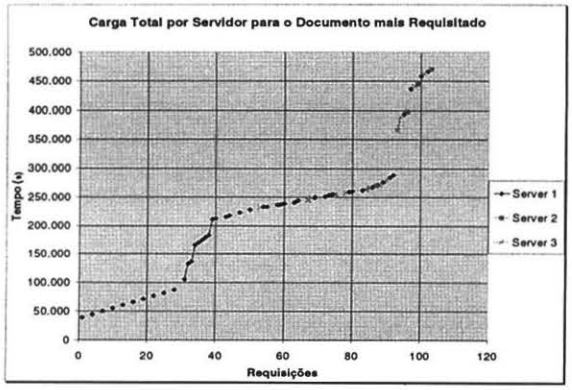

figura 5. Carga total por servidor pelo documento mais requisitado

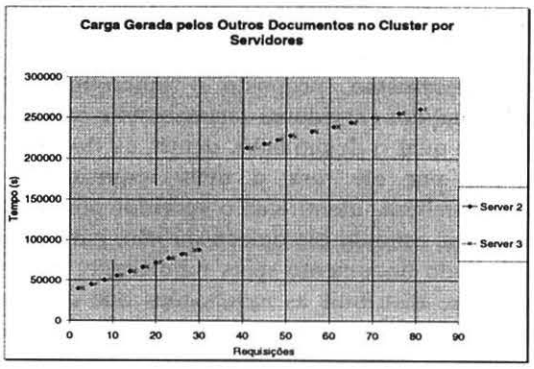

figure 6. Load Generated for Others Documents in Cluster by Servers.

7. CONCLUSÕES

Os resultados obtidos com os testes realizados, conforme apresentado no item anterior, são sintetizados nas tabelas a seguir:

TABELA 01. DISTRIBUIÇÃO DA CARGA DO TESTE 1 PELOS SERVIDORES.

TESTE 1			
Servidor	Total de Requisições Atendidas	Requisições Atendidas (Outros)	Requisições Atendidas (Doc. 3)
Agata-01	46	0	46
Agata-02	24	21	3
Agata-03	33	20	13
Totais	103	41	62

No teste apresentado, temos um total de 103 requisições distribuídas conforme mostra a tabela 01, onde pode-se notar que mais da metade das requisições geradas foram direcionadas para o documento 3 e que mesmo quando os outros servidores atendiam requisições para seus documentos, ou seja, não estavam ociosos, receberam uma parte das requisições enviadas para este documento, após sua replicação.

Em todos os testes realizados, o sistema apresentou um comportamento regular, de acordo com o problema proposto.

Este trabalho abordou a implementação de um protótipo para balanceamento de carga, por replicação de dados sob demanda, em um SGBDD baseado em *cluster*. Os resultados apresentados pelo protótipo mostraram que o balanceamento de carga entre os servidores foi realizado de forma eficiente, não dependendo nem da carga gerada, nem do servidor especificamente testado.

Constatou-se que, mesmo quando os servidores estavam sujeitos a alguma carga, estimulando um deles com a geração de uma sobrecarga, os outros servidores, após a replicação dos documentos mais

requisitados, conseguiram assumir uma parte da carga excedente do servidor testado, provendo com isso um melhor equilíbrio entre os mesmos.

O mecanismo proposto determinou com exatidão qual dos servidores estava sujeito ao excesso de carga, e qual o documento, dentre os documentos fornecidos por ele, era o mais requisitado. Foi possível, também, identificar o servidor possuindo a menor carga, estando ele ocioso ou não, para receber uma cópia do documento mais requisitado. Também conseguiu-se distribuir as requisições que chegavam para o documento replicado, os servidores que o possuíam e, portanto, balancear a carga entre eles.

Com relação a outros algoritmos, como o proposto por Cheng Fu [10], o proposto neste trabalho apresenta uma vantagem que é escolher qual o melhor documento para ser replicado, apesar do algoritmo adotado não considerar o espaço em disco no cálculo da disponibilidade de cada servidor.

O algoritmo proposto por Gwertzman [3] não pode ser empregado em *clusters* como o adotado neste trabalho, pelo simples motivo, que ao replicar o documento, ele o faz para o servidor mais próximo de onde se originaram as requisições, sem considerar a carga deste servidor que irá receber a réplica.

Quanto as possíveis aplicações deste mecanismo pode-se destacar: servidores multimídia dentro da área de educação, disseminação de informações, entretenimento (vídeo e áudio) ou em aplicação como *data mining*, que apresentam um quadro bastante similar ao abordado neste trabalho, em relação ao balanceamento de carga.

AGRADECIMENTOS

Os autores agradecem a Informix do Brasil pelo acesso ao *Informix-DS LINUX Edition 7.30.UC5*,. e pela colaboração técnica, e ao LSI-USP e a FINEP do Brasil pelo suporte financeiro.

REFERÊNCIAS

[1] Jim Gray, Gordon Bell, Parallel Database Systems 101. *VLDB 95*, Zurich, Switzerland, 1995.

[2] Jim Gray, Pat Helland, Patrick O'Neil, Dennis Shasha; The Dangers of Replication and a Solution. ACM-SIGMOD 1996, Conferencia Internacional de Gerenciamento de Dados, Quebec, 1996 pag. 173 – 182.

[3] Senior Thesis by James S. Gwertzman; Autonomous Replication In Wide-Area Internetworks, *Computer Science of Harward College, Cambridge Massachusets*, 1995.

[4] Ouri Wolfson, Sushil Jajodia, Yixiu Huang; An Adaptive Data Replicationn Algorithm, *University of Illinois and NASA/CESDIS Goddard Space Flight Center* - 1997.

[5] Gregory F. Pfister; In Search of Clusters – Norris Parker Smith, HPCwire (second edition - 1998) capítulos 1, 4 e 10.

[6] Chengzhong Xu – Nearest Neighbor Algorithms for load balancing in parallel computer – *Dept. of Electrical & Computer Eng. of Wayne State University Detroit* - 1996.

[7] Azer Bestavros – Speculative Data Dissemination and Service to reduce Server Load, Network Traffic and Service Time in Distributed Information Systems – *Computer Science Department of Boston University* - 1997.

[8] Azer Bestavros – Load Profiling A Methodology for Scheduling Real-Time Tasks in a Distributed System – *Computer Science Department of Boston University* - 1996.

[9] Master Thesis of Anastasia Anastasiadi – "A Study of Microeconomic Algorithms for Load Balancing and Data Replication in Distributed Computer Systems" - *Dept. of Computer Science University of Grete* – 1996.

[10] ChengFu Chou, Leana Golubchik – A Performance Study of Dinamic Replication Techniques in Continuous Media Servers – *Write Paper – Dept. of Computer Science University of Maryland at College Park* – 1998.

[11] Valeria Cardellini (Roma University), Michele Colajanni (Modena University), Philip S. Yu (Watson Research Center) – Dynamic Load Balancing on Web-Server Systema -Write Paper, *IEEE Internet Computing magazine* – may and june, 1999.

[12] Sergio Takeo Kofuji, Martha Ximena Torres Delgado, Edward David Moreno Ordoñez, João Antônio Zuffo - Efeito da Migração de Páginas no SPADE - I, Um Multiprocessador de Larga Escala com Memória Compartilhada - *Laboratório de Sistemas Integráveis, Escola Politécnica da Universidade de São Paulo* – 1996.