Servidores Cache WWW em Arquiteturas Multiprocessadas

Goedson Teixeira Paixão[1], Wagner Meira Jr.[1],Fernando Caixeta Sanches[1]

[1] Departamento de Ciência da Computação
Universidade Federal de Minas Gerais
Caixa Postal: 702 CEP 30123-970
Belo Horizonte - MG - Brasil
{gopaixao,meira,caixeta}@dcc.ufmg.br

Resumo—

O grande crescimento em popularidade da *World Wide Web* tem motivado várias pesquisas com o objetivo de reduzir a latência observada pelos usuários. Os servidores cache têm se mostrado uma ferramenta muito importante na busca desse objetivo.

Embora a utilização de servidores cache tenha contribuído para diminuir o tráfego na Internet, as estratégias de cooperação utilizadas na composição de grupos (*clusters*) de caches normalmente trazem uma degradação de desempenho aos servidores não sendo, por isso, escaláveis o suficiente para acompanhar o crescimento atual da WWW.

Neste trabalho, propomos uma nova forma de cooperação entre servidores cache que não cause um impacto tão grande no seu desempenho, permitindo, assim, a criação de grupos de servidores cache que sejam capazes de crescer junto com a demanda dos seus usuários.

Keywords— **WWW Cache, SMP, SHM**

I. Introdução

A utilização de servidores cache provou ser uma técnica útil na redução da latência percebida pelo usuário final da *World Wide Web*. O conceito fundamental no qual se baseia o funcionamento desses servidores é a replicação de objetos populares da Web em um local mais próximo do usuário. Os benefícios do uso de caches advêm da repetição de acessos a um determinado objeto. Se uma cópia do objeto é armazenada no servidor cache na primeira vez em que ele é acessado, os próximos acessos podem ser satisfeitos pela cópia local, não sendo necessário recorrer ao servidor original. Por exemplo, a maioria dos programas de navegação na Web possuem caches locais em disco porque em geral um usuário acessa um pequeno conjunto de objetos freqüentemente. Da mesma forma, há repetição de objetos acessados pelos membros de um dado grupo de usuários. Estes usuários poderiam se beneficiar da replicação desses objetos em um cache compartilhado em algum ponto da rede próximo deles (seu provedor de acesso à Internet, por exemplo).

Os caches WWW são implementados como agentes intermediários entre o usuário final e os servidores Web. Estes agentes armazenam localmente os objetos acessados mais recentemente para que não seja necessário recorrer ao servidor original em uma próxima requisição. Embora a utilização de caches tenha contribuído para diminuir o tráfego na Internet, as estratégias de cooperação utilizadas na composição de grupos (*clusters*) de caches normalmente trazem uma degradação de desempenho aos servidores [MFAM98a, MFAM98b] não sendo, por isso, escaláveis o suficiente para acompanhar o crescimento atual da WWW.

Neste trabalho propomos a utilização de memória compartilhada para realizar a comunicação entre servidores cache, a fim de minimizar o custo de processamento associado à cooperação entre eles, permitindo, assim, a criação de grupos de servidores cache que sejam capazes de crescer junto com a demanda de seus usuários.

Este artigo está estruturado da seguinte forma: na próxima seção apresentamos alguns trabalhos relacionados, na seção III descrevemos a estratégia de cooperação proposta, na seção IV descrevemos o ambiente que utilizamos nos testes, na seção V comentamos os resultados obtidos e na seção VI concluimos e apresentamos direções de trabalho futuras.

II. Trabalhos relacionados

O primeiro protocolo criado para cooperação entre servidores cache foi o *Internet Cache Protocol*(ICP)[WC97]. Este protocolo consiste em o servidor, ao receber uma requisição de um objeto que ele não possui, perguntar a cada um de seus parceiros se algum deles possui esse objeto e recuperá-lo daquele que o possuir.

Com o objetivo de eliminar os custos de comunicação existentes no protocolo ICP, foi introduzida na versão 2 do servidor cache Squid a cooperação através de *Cache Digests*[RW98]. Esta estratégia de cooperação consiste em os servidores enviarem, periodicamente, resumos do conteúdo dos seus próprios caches para os seus parceiros. Desta forma os parceiros podem consultar a sua cópia do resumo, evitando assim a troca de mensagens a cada requisição atendida.

A Microsoft introduziu, em 1997, no Microsoft Proxy Server 2.0, o *Cache Array Routing Protocol*[Mic98], um protocolo para cooperação entre servidores cache que permite a qualquer servidor cache, descobrir, sem a necessidade de

comunicação, qual servidor possivelmente possui um determinado objeto. Este protocolo baseia-se no particionamento do espaço de URLs entre os servidores parceiros, de forma que apenas um deles possa ter uma cópia local de um determinado objeto.

Apesar de procurarem uma forma de colaboração mais eficiente entre os servidores, todos esses protocolos utilizam o protocolo HTTP para fazer a transferência de objetos entre os servidores ignorando a possibilidade de existência de meios de comunicação mais eficientes.

III. SHMSquid

As estratégias para cooperação entre servidores cache existentes atualmente utilizam o protocolo HTTP para transferência de documentos entre os servidores cooperantes. Isto impede que os servidores possam tirar proveito dos recursos existentes em instalações mais específicas como redes locais de alta velocidade ou máquinas multiprocessadas. Nesta seção descrevemos uma nova estratégia de cooperação entre servidores cache, projetada com o objetivo de eliminar os custos associados à troca de informações entre os servidores. Essa estratégia utiliza as facilidades de comunicação disponíveis para processos que estejam executando em uma mesma máquina ou ambientes de memória compartilhada distribuída (DSM)[MLHA97, BKP+96].

A. *Cooperação entre caches usando memória compartilhada*

A disponibilidade de máquinas com arquitetura SMP a baixo custo vem crescendo de maneira acelerada, tornando viável (e possivelmente mais barato devido ao compartilhamento de vários componentes) a substituição de um grupo de servidores cache composto por uma rede de máquinas de pequeno porte por uma máquina dotada de vários processadores semelhantes aos das máquinas originais, com a vantagem de uma comunicação mais eficiente entre servidores executando nesta nova configuração (em comparação com servidores executando em máquinas separadas na rede original). Entretanto, os servidores cache existentes não são capazes de tirar proveito desta nova capacidade de comunicação porque os seus métodos de cooperação são baseados no protocolo TCP/IP, não importando o meio de comunicação utilizado e ignorando recursos que possam estar sendo compartilhados pelos processos.

A fim de aproveitar as facilidades de comunicação disponíveis para processos que estejam executando em uma máquina SMP, implementamos no servidor cache Squid[Nat] um novo método de cooperação que permite que um servidor recupere dados existentes no cache de um parceiro sem interferir no processamento deste último. Esta nova estratégia funciona da seguinte forma (uma descrição mais detalhada pode ser encontrada em [Goe99]):

1. Cada servidor cria um espaço de memória compartilhado no qual coloca a sua tabela de `StoreEntrys`[1];
2. Ao receber uma requisição, o servidor procura na sua própria tabela de `StoreEntrys` pela existência do objeto desejado;
3. Caso não encontre o objeto desejado, o servidor procura na tabela de cada um de seus parceiros pela existência do objeto;
4. Se o objeto é encontrado na tabela de algum dos parceiros, o servidor recupera o objeto do arquivo contido no diretório de cache do seu parceiro e o envia para o cliente que fez a requisição.

Fazendo a recuperação de objetos existentes no cache de parceiros desta forma, evitamos os custos de comunicação e eliminamos as várias chamadas de sistema feitas para troca de dados entre os servidores tornando o custo deste atendimento igual ao de um atendimento satisfeito pelo cache local.

A principal desvantagem deste método, com relação aos métodos já existentes, vem do fato de o parceiro não participar na transação e, portanto, não contabilizar o acesso feito. Como o acesso feito via memória compartilhada não é contabilizado pelo servidor que controla aquele objeto, a sua lista de objetos utilizados mais recentemente não fica corretamente atualizada resultando na eliminação de objetos acessados recentemente pelos seus parceiros (mas não por ele próprio) conservando objetos menos acessados ocasionando uma taxa de acerto global mais baixa. Este problema, no entanto, pode ser resolvido pela implementação de um mecanismo de comunicação entre os servidores que torne possível informar ao parceiro sobre a utilização de objetos do seu cache.

B. *Comunicação através de memória compartilhada*

A variação System V dos sistemas operacionais UNIX introduziu três mecanismos de comunicação entre processos que estão atualmente disponíveis na maioria das versões de UNIX existentes no mercado. Entre esses mecanismos de comunicação está o uso de memória compartilhada.

A implementação de memória compartilhada no System V permite que um processo compartilhe uma região do seu espaço de endereçamento e atribua a essa região permissões de acesso para leitura e/ou escrita, discriminando entre processos do mesmo usuário, processos de usuários do mesmo grupo ou processos de usuários de outros grupos. Um outro processo pode, então, anexar essa região de memória ao seu próprio espaço de endereçamento e acessar os dados nela contidos utilizando as mesmas instruções que usaria para acessar qualquer outra parte da sua memória (obedecendo as restrições de acesso estabelecidas pelo processo criador). Maiores detalhes sobre a utilização de memória compartilha-

[1] Conjunto de dados que o Squid mantém em memória principal sobre cada um dos objetos contidos no seu cache.

da em sistemas UNIX podem ser encontrados em [Bac86].

Para facilitar a utilização de memória compartilhada no Squid implementamos um módulo que nos permite criar um segmento de memória compartilhada e gerenciar esse espaço fornecendo uma interface para alocação e liberação de memória semelhante às funções `calloc` e `free` da biblioteca C padrão. Esse módulo fornece as seguintes funções para manipulação da memória compartilhada:

- `int shm_init(key, size)` - Esta função cria um segmento de memória associado à chave `key` com tamanho suficiente para conter um apontador mais `size` bytes. O número de bytes disponíveis para alocação no segmento de memória inicializado por `shm_init` é igual à menor potência de 2 que seja maior ou igual ao valor `size`. O valor de retorno é igual a 0 em caso de sucesso e o endereço no qual o segmento de memória compartilhada foi anexado é colocado na primeira palavra do segmento para permitir que outros processos traduzam apontadores que possam estar contidos nesse segmento para o seu próprio espaço de endereçamento. Esta função cria também as listas de blocos de memória livres e alocados utilizadas no gerenciamento do espaço de memória compartilhada.
- `void *shm_connect(key)` - Esta função anexa o segmento de memória associado com a chave `key` ao espaço de endereçamento do processo em modo de somente leitura e retorna o apontador para a área onde o segmento foi anexado.
- `shm_disconnect(mem)` - Esta função retira do espaço de endereçamento do processo o segmento de memória apontado por `mem`. Caso o segmento tenha sido marcado para remoção e não haja mais processos o utilizando, ele será liberado pelo sistema.
- `shm_clear(mem)` - Esta função marca o segmento de memória apontado por `mem` para ser removido quando não houver mais processos que o estejam utilizando.
- `SHARED_PTR(base, ptr)` - Esta macro permite a um processo traduzir endereços contidos no segmento de memória compartilhada do espaço de endereçamento do processo criador do segmento para o seu próprio espaço de endereçamento de forma a poder referenciar a memória apontada por ele. O parâmetro `base` é o endereço em que o segmento foi anexado no espaço de endereçamento do processo e `ptr` é o apontador a ser traduzido. Esta macro utiliza o fato de o segmento de memória conter em sua primeira posição o endereço em que ele começa na memória do processo criador.

C. *Implementação da cooperação usando memória compartilhada no Squid*

Neste trabalho, fizemos a implementação de um mecanismo de colaboração entre servidores cache que permite que um processo recupere objetos existentes no cache de um outro processo sem a intervenção deste último, buscando reduzir o custo de colaboração entre servidores que estejam sendo executados em uma mesma máquina.

Nesta seção descrevemos os pontos principais da implementação deste mecanismo de colaboração.

C.1 Configuração e compartilhamento dos dados

Para permitir o compartilhamento de dados entre os servidores Squid através de memória compartilhada, adicionamos os seguintes parâmentros ao arquivo de configuração:

- `shm_key key` - Este é um parâmetro obrigatório no SHMSquid. Ele determina que `key` será a chave utilizada para criação do segmento de memória compartilhada. Este valor será usado pelos parceiros existentes na mesma máquina para acessar os dados compartilhados pelo servidor. O parâmetro `key` deve ser um valor inteiro.
- `shm_size size` - Este parâmetro determina o tamanho do segmento de memória compartilhada a ser utilizada pelo servidor. Este valor deve ser grande o suficiente para todas as `StoreEntrys` que serão criadas durante o seu funcionamento. Este parâmetro pode ser omitido, sendo assumido o valor padrão de 8 MB.
- `shm_peer key` - Este parâmetro é utilizado para indicar a chave utilizada por outro servidor para alocar a sua memória compartilhada, permitindo ao Squid acessar os dados que esse servidor colocou nessa área. Deve haver uma entrada `shm_peer` para cada servidor com o qual queiramos colaborar via memória compartilhada e o valor `key` deve ser igual ao valor especificado por `shm_key` na configuração dos outros servidores.

A estrutura de configuração `Config` foi modificada para conter os novos dados de configuração sendo que o campo `shm_peers` contém uma lista de parceiros contendo, para cada um deles, um apontador para a sua tabela de `StoreEntrys`, a sua configuração de diretórios de cache e o endereço onde se inicia a memória compartilhada daquele servidor.

Após a leitura do arquivo de configuração, o Squid cria o segmento de memória compartilhada definido no arquivo de configuração (usando a função `shm_init`) e aloca no início desse segmento uma estrutura contendo dois apontadores: um para a cópia das suas configurações de diretórios cache colocada na memória compartilhada e outro para a tabela *hash* contendo as suas `StoreEntrys` (que passam a ser todas alocadas através da função `shm_calloc`).

Para que um processo possa distinguir as suas próprias `StoreEntrys` das criadas por outros processos, acrescentamos à estrutura da `StoreEntry` o campo `owner_key` onde o criador da entrada armazena a chave definida por `shm_key` no arquivo de configuração. Desta forma, o servi-

dor é capaz não só de saber que aquela entrada pertence a outro processo como também identificar na sua configuração os dados necessários para recuperar o objeto a que a entrada se refere.

C.2 Recuperação de um objeto em cache de outro servidor

Nesta seção iremos descrever o mecanismo de recuperação de objetos que se encontram no cache de servidores que estão configurados para colaborar através da utilização de memória compartilhada.

O servidor SHMSquid coloca todas as suas `StoreEntrys` na região de memória compartilhada, de forma que um outro processo possa consultar o conteúdo do seu cache sem que, para isso, seja necessário interferir no seu processamento. Para evitar condições de corrida que poderiam surgir no caso de haver um compartilhamento total desses dados (com processos podendo modificar as `StoreEntrys` de outros processos), decidimos que um servidor acessaria a memória de outros em modo de somente leitura.

Como mencionado na seção C, a maior parte do atendimento feito utilizando dados obtidos via memória compartilhada é semelhante ao atendimento de uma requisição usando dados do próprio servidor. Aqui iremos descrever os pontos em que estes atendimentos diferem.

Durante o atendimento normal, o Squid cria uma estrutura `MemObject` associando-a à `StoreEntry` encontrada. A estrutura `MemObject` serve para o Squid controlar o processo de envio do objeto e contém dados como: o tamanho do objeto sendo transmitido, quanto está em memória (já foi lido do disco) e quanto já foi enviado. Mas, para fazer o atendimento utilizando os dados de outro processo, não podemos modificar a `StoreEntry` e, por isso, não podemos fazer a associação do `MemObject` como o Squid faz normalmente. Desta forma criamos uma nova estrutura chamada `ShmHitCallbackData` que utilizamos durante o atendimento. Essa estrutura contém apontadores para a `StoreEntry` encontrada, o `MemObject`, a requisição sendo atendida e a configuração de diretórios do servidor que controla o objeto desejado.

Como a maior parte das rotinas utilizadas no atendimento a uma requisição recebe uma `StoreEntry` como parâmetro e acessa os outros dados necessários (`MemObject`, por exemplo) através dela, precisamos implementar novas rotinas que recebem a `ShmHitCallBackData` como parâmetro e fazem acessos aos dados necessários através dessa estrutura. Estas rotinas, no entanto, foram implementadas seguindo a mesma filosofia de atendimento utilizada no tratamento de um *hit*. Desta forma, apesar de o atendimento a um *hit* local e um *hit* em memória compartilhada serem realizados por conjuntos diferentes de rotinas, eles são feitos de maneira semelhante.

Como esta nova estratégia não traz custos adicionais devidos à cooperação entre os servidores cache, esperamos que ela venha a ser uma opção mais eficiente na implementação de servidores cache escaláveis.

IV. Ambiente experimental

Nos experimentos realizados durante este trabalho, utilizamos a metodologia para análise de desempenho de hierarquias de servidores cache sugerida em [Fon99].

Utilizamos como servidor cache um PC dotado de 4 processadores PentiumPro de 200MHz e 128Mb de memória.

As requisições de objetos foram geradas por quatro estações SPARCstation SLC, utilizando uma carga de trabalho baseada em *logs* do servidor cache do POP-MG[MdSFAM98]. Analisamos um *log* que contém 4,235,511 requisições para 1,079,044 objetos diferentes totalizando aproximadamente 12 Gb de dados. Para os nossos experimentos, geramos um conjunto de 80,000 requisições baseadas na análise desse *log*, totalizando 400Mb de dados. Este conjunto de requisições é dividido entre os clientes, de forma que cada um deles gera uma seqüência de 20,000 requisições HTTP para os servidores cache.

Como servidores HTTP, utilizamos 3 estações SPARCstation SLC executando, cada uma delas, um processo que faz o atendimento de requisições HTTP gerando um arquivo baseado na URL pedida e introduzindo um atraso na resposta proporcional ao tamanho desse arquivo.

Todas as estações SPARCstation SLC estão ligadas aos servidores cache através de uma rede Ethernet de 10Mbps chaveada.

V. Análise de resultados

Para verificar a escalabilidade da estratégia de cooperação proposta, realizamos experimentos utilizando um número variável (entre 2 e 4) de servidores SHMSquid tendo sempre um espaço total em disco de 100MB para armazenamento dos objetos do cache. A Tabela I mostra as configurações de cada um dos experimentos realizados. A Tabela II mostra o tempo de atendimento médio de uma requisição e a Tabela III mostra a taxa de acerto no cache.

Tabela I
CONFIGURAÇÃO DOS EXPERIMENTOS
Disco é o espaço em disco para cada servidor e **Clientes** é o número de conexões simultâneas em cada servidor

Configuração	Disco	Clientes
shm2	50Mb	40
shm3	33Mb	26
shm4	25Mb	20

Analisando os dados da Tabela II, podemos ver que a

Tabela II

LATÊNCIA (SEGUNDOS)

Cliente é o tempo médio de uma requisição medido pelo cliente, **Hit** e **Miss** são os tempos médios para servir um *hit* e um *miss*, respectivamente, medidos pelo servidor

Configuração	Cliente	Hit	Miss
shm2	5.2863	0.0037	0.0056
shm3	5.3622	0.0037	0.0056
shm4	5.9948	0.0037	0.0056

Tabela III

TAXA DE ACERTO NO CACHE

Local é a porcentagem de requisições que foram servidas com dados do próprio servidor e **Cooperação** é a porcentagem das requisições que foram atendidas utilizando dados dos caches dos parceiros

Configuração	Local	Cooperação	Total
shm2	0.3730	0.1095	0.4825
shm3	0.3270	0.1396	0.4670
shm4	0.3442	0.0922	0.4367

cooperação através de memória compartilhada não impõe custos devido ao acesso aos dados dos parceiros, pois os tempos médios para atendimento de *hits* e *misses* se mantêm os mesmos nas três configurações.

Apesar de a colaboração através de memória compartilhada não causar um aumento no tempo de atendimento de cada requisição, podemos observar um aumento no tempo médio observado pelos clientes à medida que aumentamos o número de servidores. Analisando os dados da Tabela III, podemos observar que esse aumento está relacionado a uma menor taxa de acerto no cache que, como mencionamos anteriormente, é causado, em parte, pela falta de sincronização entre os servidores. Outro fator causador da diminuição da taxa de acerto é a fragmentação do espaço em disco que, nas configurações com mais servidores, fica dividido em partes menores (como ilustrado na Tabela I), de forma que o espaço livre pode estar divido entre as várias partes impossibilitando a inclusão de um objeto que poderia ter sido armazenado caso o espaço em disco fosse contíguo.

VI. CONCLUSÕES E TRABALHOS FUTUROS

Pudemos verificar que a cooperação entre servidores cache através do uso de memória compartilhada mostra-se como uma boa alternativa para a implementação de grupos de servidores cache em arquiteturas multiprocessadas.

Esta estratégia de cooperação, no entanto, leva a uma taxa de acerto menor devido à falta de sincronização entre os servidores. Este problema pode ser resolvido através da implementação de mecanismos de comunicação entre os servidores que possibilitem a um servidor comunicar ao seu parceiro que um objeto do seu cache está sendo utilizado.

REFERÊNCIAS

[Bac86] Maurice J. Bach. *The Design of the UNIX operating system*. Prentice-Hall Software Series. P T R Prentice-Hall, Inc., 1986.

[BKP+96] R. Bianchini, L. Kontothanassis, R. Pinto, M. De Maria, M. Abud, and C. Amorim. Hiding communication latency and coherence overhead in software DSMs. In *Proceedings of the 7th International Conference on Architectural Support for Programming Languages and Operating Systems*. Boston,MA, October 1996.

[Fon99] Erik Luiz da Silva Fonseca. Hierarquia de Servidores Proxy Cache WWW: Instrumentação e Análise de Desempenho. Master's thesis, Universidade Ferderal de Minas Gerais, 1999.

[Goe99] Goedson Teixeira Paixão. Escalabilidade em servidores cache WWW. Master's thesis, Universidade Ferderal de Minas Gerais, 1999.

[MdSFAM98] Wagner Meira Jr., Erik Luiz da Silva Fonseca, Virgílio Augusto Fernandes Almeida, and Cristina D. Murta. Performance Analysis of WWW Cache Proxy Hierarquies. *Journal of the Brazilian Computer Society*, 5(2), Novembro 1998.

[MFAM98a] Wagner Meira Jr., Erik Fonseca, Virgílio Almeida, and Cristina Murta. Analyzing performance of cache server hierarchies. In *Proceedings of SCCC'98*, Antofagasta, Chile, November 1998.

[MFAM98b] Wagner Meira Jr., Erik Fonseca, Virgílio Almeida, and Cristina Murta. Evaluating and configuring WWW caches. In *3rd International WWW Caching Workshop*, Manchester, UK, June 1998.

[Mic98] Microsoft Corporation. *Cache Array Routing Protocol and Microsoft Proxy Server 2.0*, 1998. `http://www.microsoft.com/proxy/documents CarpWP.exe`.

[MLHA97] W. Meira Jr., T. J. LeBlanc, N. Hardavellas, and C. Amorim. Understanding the performance of DSM applications. In *Proceedings of the Workshop on Communication and Architectural Support for Network-based Computing (CANPC)*, volume 1199 of *Lecture Notes in Computer Science*, pages 98–211, San Antonio, TX, February 1997. IEEE, Springer-Verlag.

[Nat] National Laboratory for Applied Network Research. *Squid Internet Object Cache*. http://squid.nlanr.net/Squid/.

[RW98] Alex Rousskov and Duane Wessels. Cache digests. `http://ircache.nlanr.net/Papers/ cache-digests.ps.gz`, April 1998.

[WC97] Duane Wessels and K. Claffy. Internet cache protocol(icp), version 2. Network Working Group RFC 2186, September 1997. `http://ds.internic.net/rfc/rfc2186.txt`.