eCache: uma Cache Cooperativa para Servidores de Comércio Eletrônico Baseados em Cluster

Silvano Dias[1], Anderson Silva[1], Thobias Trevisan[1], Wagner Meira Jr.[2], Claudio Amorim[1]

[1] Laboratório de Computação Paralela – COPPE Sistemas
Universidade Federal do Rio de Janeiro, RJ – Brasil
{silvano, faustino, thobias, amorim}@cos.ufrj.br

[2] e-SPEED - Departamento de Ciência da Computação
Universidade Federal de Minas Gerais, Belo Horizonte, MG – Brasil
meira@dcc.ufmg.br

Resumo

A popularidade do comércio eletrônico na WWW tem aumentado a demanda sobre os servidores, degradando seu desempenho e exigindo assim, novas soluções para assegurar sua qualidade de serviço. Normalmente um dos principais limitadores do desempenho e escalabilidade do servidor é o gerenciador de banco de dados. Neste trabalho, introduzimos a eCache, uma cache cooperativa no nível da aplicação que combina os modelos de programação de memória compartilhada e de passagem de mensagens, a fim de diminuir o número de acessos ao banco de dados e melhorar a escalabilidade de servidores de comércio eletrônico baseados em cluster. Nós avaliamos os potenciais benefícios da eCache medindo o desempenho de um protótipo de livraria virtual, submetido a uma carga de trabalho gerada a partir de um log real. Nossos resultados mostram que um servidor com 8 computadores utilizando eCache pode melhorar seu tempo de resposta e sua taxa de conexões entre 8 e 10 vezes comparado a um servidor sem cooperação de cache. Sobretudo, estes resultados preliminares sugerem que a eCache oferece um mecanismo promissor com o qual é possível se implementar servidores escaláveis em clusters para aplicações de comércio eletrônico.

1. Introdução

Nos últimos anos, a popularidade do comércio eletrônico [12] na Internet tem imposto uma demanda crescente aos servidores, degradando seu desempenho e exigindo novas soluções escaláveis para assegurar a qualidade de serviço que os consumidores normalmente esperam. É de conhecimento geral que os clientes são freqüentemente suscetíveis ao tempo de resposta de *sites* e tendem a abandonar aqueles em que experimentam demoras indesejáveis, trocando para outros a procura de melhor qualidade de serviço. Por isso, é essencial que os servidores tenham desempenho escalável, mantendo um tempo de resposta adequado, para acompanhar a demanda crescente de clientes sobre as aplicações de comércio eletrônico.

Atualmente, clusters de computadores pessoais conectados por uma rede de alta velocidade estão se tornando a plataforma de hardware preferida para servidores de alto desempenho, devido ao seu custo-benefício. Normalmente, um servidor de comércio eletrônico baseado em cluster utiliza um de seus computadores como o servidor WWW, que recebe os pedidos dos clientes e os distribui para os servidores de transações presentes nos demais computadores. Estes por sua vez, são responsáveis pelo processamento dos pedidos, obtendo as informações necessárias do servidor de banco de dados e enviando as respostas de volta ao servidor WWW, que as repassa aos clientes. Entretanto, alcançar desempenho escalável em tal plataforma não é uma tarefa simples. Como resultado disso, vários mecanismos têm sido propostos para distribuição de requisições e balanceamento de carga em servidores WWW [1, 4, 8]. Em particular, políticas de distribuição baseadas na análise do conteúdo das requisições mostram que é possível atingir boa localidade na cache dos servidores de transações e, assim, melhorar a vazão do servidor WWW [2, 15].

Nosso trabalho é complementar a estes no sentido de que nosso enfoque principal é investigar o impacto da limitada taxa de processamento do gerenciador de banco de dados no desempenho de servidores de comércio eletrônico. Em particular, fomos motivados pela observação de que aplicações típicas de comércio eletrônico apresentam intensa manipulação de dados, tornando o gerente de banco de dados uma fonte potencial de degradação do desempe-

nho. Normalmente, pedidos dos clientes geram complexas pesquisas por dados, causando vários acessos de alto custo a discos magnéticos que prejudicam a vazão do servidor.

Neste artigo introduzimos a eCache, uma cache cooperativa no nível da aplicação que implementa uma única cache distribuída e compartilhada sobre as caches locais dos servidores de transações. Basicamente, sempre que um servidor de transações consulta o banco de dados, a eCache torna o dado visível e compartilhável entre todos os computadores. Deste modo, ela permite o servidor de transações eliminar acessos redundantes ao banco de dados para dados que já estão armazenados na cache local de algum outro.

A organização da eCache divide a cache de cada servidor de transações em dois segmentos: 1) o segmento local, que armazena os dados da aplicação e 2) o segmento global, que implementa um diretório de meta-dados sobre tudo que está disponível nos segmentos locais. Nossa implementação emprega modelos de programação paralela diferentes para cada segmento. Utilizamos passagem de mensagens sobre *sockets* TCP/IP para o segmento local, uma vez que este envolve transferências de grandes quantidades de dados. E para o segmento global, que manipula uma quantidade menor de dados, foi usada memória compartilhada (DSM) através do software HLRC [11]. O uso de DSM simplificou bastante a implementação da cooperação, pois as operações de *lock* e *unlock* oferecem um modo simples e fácil de garantir a consistência dos dados no segmento global.

Nós avaliamos o impacto da eCache no desempenho de um livraria virtual em um cluster de 14 computadores Pentium III 650 MHz, com 512 MB de memória RAM cada, conectados por um *switch* Fast Ethernet de 100 Mbps e um *switch* Giganet de 1,25 Gbps. Foram utilizados os softwares Apache como o servidor WWW e MySQL como o gerenciador de banco de dados. Nossos experimentos foram baseados em uma carga de trabalho com mais de 65.000 requisições de pesquisa obtidas de um *log* real de um servidor de comércio eletrônico. Para aplicar a carga foi usado o *benchmark* WebStone, que simulou 16 clientes em 4 computadores. Os resultados obtidos mostram a escalabilidade do servidor até 8 computadores executando o protótipo da livraria virtual. Com o auxílio da eCache, o tempo de resposta e a taxa de conexões melhoraram significativamente, alcançando valores de 8 a 10 vezes melhores do que o servidor sem cooperação de cache. Em suma, este artigo apresenta as seguintes contribuições:

1. Introdução da eCache, uma nova cache cooperativa no nível da aplicação de comércio eletrônico para servidores baseados em cluster.

2. Descrição da implementação eficiente de uma livraria virtual baseada nos modelos de programação por passagem de mensagens e memória compartilhada.

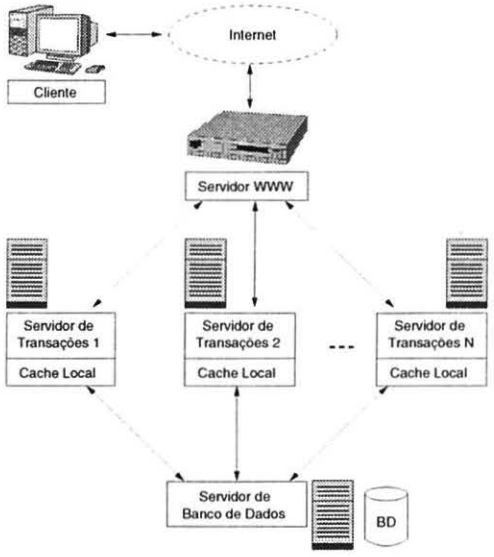

Figura 1. Componentes de um servidor de comércio eletrônico

3. Avaliação de desempenho do uso da eCache em um servidor escalável de comércio eletrônico.

O restante do texto está organizado em mais seis seções. Na seção 2 apresentamos a estrutura básica de um servidor de comércio eletrônico. A seção 3 descreve o projeto e implementação da eCache. Na seção 4 apresentamos nossa metodologia de testes e resultados obtidos. Na seção 5 comparamos nossa implementação com trabalhos relacionados. E a seção 6 finaliza com nossas conclusões.

2. Servidores de Comércio Eletrônico

Um servidor de comércio eletrônico difere de um servidor WWW convencional por fornecer funcionalidades e serviços adicionais. De fato, um servidor WWW é um dos componentes de um servidor de comércio eletrônico. A principal diferença entre os dois é que durante as requisições a aplicação de comércio eletrônico armazena certos dados da interação com cada cliente, necessários às transações que estão sendo efetuadas por cada um deles. Esta funcionalidade, chamada sessão de usuário, é importante em comércio eletrônico, pois sem ela as transações não poderiam ser compostas por mais de uma requisição.

Um servidor de comércio eletrônico baseado em cluster consiste de vários computadores comuns, ou estações de trabalho, conectados por uma rede de alta velocidade. A Figura 1 mostra a estrutura básica do servidor com seus três componentes principais:

1. Servidor WWW, que implementa a interface através da

qual os clientes podem interagir com o servidor. Basicamente, ele é o gerente de tarefas que decide qual servidor de transações deverá processar cada pedido dos clientes. Um ou mais computadores do cluster podem ser alocados para esta função a fim de melhorar a taxa de conexões do servidor.

2. Servidor de Transações, que é responsável por processar os pedidos dos clientes, tais como pesquisar e selecionar produtos, adicionar ou remover um produto do "carrinho de compras". Uma vez completada a operação, ele envia a resposta ao servidor WWW, que a repassa ao cliente. Note que a natureza do comércio eletrônico envolve, em muitos casos, atualizações freqüentes a dados que serão enviados aos clientes. Normalmente é neste componente do servidor que mais se varia o número de computadores.

3. Servidor de Banco de Dados, que armazena toda a informação sobre os produtos de uma loja virtual, como descrição e quantidade em estoque, e fornece várias funcionalidades para garantir um acesso seguro, eficiente e padronizado aos dados, como por exemplo, o uso de índice nas tabelas e controle de acesso dos usuários. Além disso, ele garante as três propriedades fundamentais para transações em comércio eletrônico [12]: atomicidade, consistência, e persistência. É comum este componente utilizar somente um ou dois computadores do cluster.

É interessante lembrar que servidores WWW baseados em cluster, considerados o estado-da-arte, incorporam mecanismos e políticas eficientes que melhoram consideravelmente o desempenho. Entretanto, o problema da limitada vazão dos gerenciadores de banco de dados ainda permanece sem solução para servidores de comércio eletrônico que possuam altas taxas de requisições. Na próxima seção nós tratamos desta questão propondo a eCache, um mecanismo de cooperação eficiente que pode aliviar significativamente a demanda sobre o servidor de banco de dados.

3. Projeto e Implementação da eCache

3.1. Organização da eCache

A principal característica no projeto da eCache é seu suporte ao compartilhamento de dados entre todos os servidores de transações. A Figura 2 mostra um esquema da sua organização, a qual divide a área de memória destinada para cache em dois segmentos: local e global. O segmento local armazena os dados utilizados pela aplicação, que no caso deste protótipo são livros, enquanto que o segmento global implementa um diretório distribuído de meta-dados,

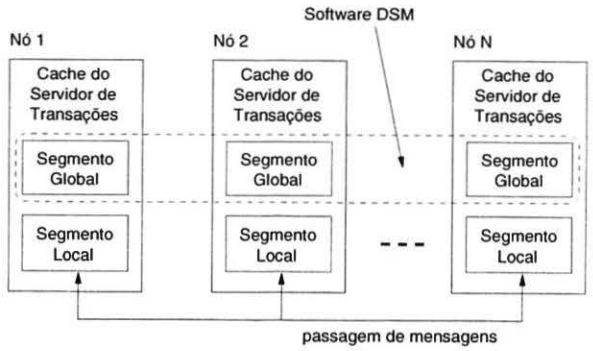

Figura 2. Segmentos da eCache

o qual permite a aplicação identificar os dados que estão disponíveis nos segmentos locais dos outros computadores.

A eCache funciona da seguinte maneira. Quando um servidor de transações precisa de algum dado, ela verifica primeiro no seu segmento local. Caso não encontre o dado desejado, ela tenta o segmento global. Se for encontrada uma entrada no segmento global, os meta-dados correspondentes indicarão quais dos outros servidores de transações têm cópias extras dos dados. Depois de pedir e obter os dados remotos, uma cópia deles é armazenada no segmento local e a informação global atualizada. Deste modo, os servidores de transações podem evitar acessos desnecessários ao banco de dados. Caso a entrada desejada também não seja encontrada no segmento global ou algum dos livros não seja encontrado em nenhum outro computador, a aplicação terá de recorrer ao servidor de banco de dados e armazená-los na eCache.

Os dados dos livros são distribuídos em três níveis dentro de cada segmento. O segmento local é organizado da seguinte maneira:

- Nível 1: contém os termos que foram pesquisados pelos clientes, cada qual com uma lista dos livros que se referem a ele.

- Nível 2: contém o ISBN, código, título e autores de cada livro pedido.

- Nível 3: guarda o restante das informação dos livros, usadas para uma visualização mais detalhada do produto, como descrição, editora, número de páginas etc.

O segmento global também possui uma organização de três níveis, embora armazene somente informações sobre a localização dos livros. Sua organização pode ser resumida como segue:

- Nível 1: guarda os termos pesquisados, cada um com uma lista de livros que contêm estas palavras.

- Níveis 2 e 3: contêm todos os ISBN dos livros cadastrados no banco de dados e um vetor de 32 bits cada, que indica quais dos servidores de transações possui uma réplica dos dados do livro nos níveis 2 ou 3 de seu segmento local.

3.2. Política de Substituição

Sempre que um servidor de transações obtém um novo dado e a eCache está cheia, seja no segmento local ou global por quantidade de bytes ou entradas, é utilizada a política de LRU para substituir os dados no segmento correspondente. Após ser descartada uma entrada global, o software DSM se encarrega da coerência dos meta-dados entre todos os servidores de transações. No caso do segmento local, a política de substituição também leva em conta a informação global e dá prioridade para remover as entradas que têm réplicas. Esta tentativa é feita sobre 1% das que foram menos recentemente utilizadas e se nenhuma possui réplicas, é removida a mais antiga mesmo. A política de substituição ocorre em duas situações:

1. Nível 1 cheio: o termo de pesquisa menos recentemente usado é removido e cada livro no nível 2 que é referenciado por ele tem seu contador de referências diminuído. Quando este contador chega a zero, o livro é removido do nível 2, mas somente se existe uma réplica em outro computador. Por causa disso, alguns livros podem permanecer no nível 2 sem serem referenciados por nenhuma entrada do nível 1.

2. Nível 2 ou 3 cheio: as entradas menos recentemente usadas no nível correspondente são descartadas até que haja espaço suficiente para armazenar os novos dados na cache. Note que sempre uma entrada do nível 3 é referenciada por uma do nível 2, ou seja, quando a do nível 2 for removida, a do nível 3 também é descartada, mas o contrário não ocorre.

No caso do segmento global, somente a primeira situação ocorre, mas a política de substituição nunca remove as entradas dos níveis 2 e 3, pois estas não possuem dados e sim informação de localização dos dados.

3.3. Questões de Implementação

3.3.1. Modelos de Programação

Nossa decisão de utilizar dois modelos diferentes de programação para implementar a eCache, em particular memória compartilhada, foi baseada em três observações chave. Primeiro, o segmento global ocupa uma área de memória relativamente pequena, possível de ser implementado com um software DSM. Segundo, o modelo de programação com memória compartilhada facilita a implementação da cooperação de cache, utilizando somente operações de *lock* e *unlock* para sincronizar os dados. E terceiro, o modelo de programação por passagem de mensagens, embora mais complexo para o desenvolvimento, é capaz de manipular quantidades maiores de dados de forma mais eficiente, pois não possui a limitação de memória que a maioria dos softwares DSM apresenta por terem de replicar a área compartilhada em todos os computadores.

3.3.2. Cooperação de Cache

A eficiência da eCache está diretamente relacionada a redução do número de acessos ao banco de dados e pode ser medida pela taxa de acertos (*hits*) na cache. Note que para maximizar esta taxa, a eCache deve oferecer o máximo possível de cooperação entre as caches locais. Por isso é que sua política de substituição tenta manter pelo menos uma cópia de cada livro em memória, mesmo que não tenha sido utilizado recentemente. Assim o servidor de transações não precisará pagar o alto preço de um acesso ao banco de dados, caso o livro seja requisitado novamente.

4. Avaliação da eCache

4.1. Ambiente de Testes

Os experimentos para avaliar a eCache foram realizados em um conjunto de 14 computadores, cada um com processador Pentium III 650 MHz e 512 MB de memória RAM, conectados por dois sistemas de rede com *switches*: um Giganet de 1,25 Gbps e um Fast Ethernet de 100 Mbps. O primeiro foi destinado ao tráfego do software DSM, com o protocolo VIA [5], enquanto que o restante da comunicação ocorreu através do segundo, com TCP/IP.

A Tabela 1 mostra os softwares e o número de computadores utilizados nos testes. Uma versão mais antiga do sistema operacional foi usada no servidor de transações e no simulador de clientes por uma razão de estabilidade, pois o *driver* VIA e o WebStone apresentaram comportamento anômalo sobre um *kernel* mais atual.

Componente	Software	Linux	Nós
Servidor WWW	Apache 1.3.22	2.4.14	1
Banco de Dados	MySQL 3.23.33	2.4.14	1
Serv. de Transações	HLRC 1.0	2.2.14	1 a 8
Clientes	WebStone 2.0.1	2.2.14	4

Tabela 1. Componentes dos testes

Ao software Apache foi adicionado um módulo, ligado estaticamente (*built-in*), para distribuir as requisições dos clientes aos servidores de transações. O método de

distribuição utilizado foi *round-robin*, onde não é feita nenhuma verificação do conteúdo das requisições e não é mantida nenhuma informação sobre carga ou cache dos nós que as processam. Entre os servidores de transações nenhuma requisição é redirecionada, ou seja, o nó que a recebeu processa, obtém os dados necessários de onde for possível, monta o resultado em HTML e o envia ao Apache.

A base de dados usada nos testes é composta de 40.000 livros que correspondem a um total de 130 MB. Cada um destes livros ocupa aproximadamente 3,35 KB para ser armazenado na cache do servidor de transações.

4.2. Configurações de Cache

Devido ao pequeno tamanho da base de dados, comparado à memória RAM disponível em cada computador, foram utilizados tamanhos reduzidos de cache para simular casos onde somente parte dos dados pudesse ser armazenada em memória.

A Tabela 2 apresenta todas as configurações de cache usadas nos experimentos. Os valores mostram a quantidade de memória gasta em cada servidor de transações. O rótulo `sc` corresponde ao servidor executando somente com o segmento local, sem cooperação de cache, e o rótulo `ec` identifica as execuções do servidor com a organização completa da eCache.

Configuração	Global nível 1	Local nível 1	Local níveis 2+3	Total
sc 16% bd	-	65	21	86
ec 10% bd	65	8	13	86
sc 26% bd	-	65	34	99
ec 20% bd	65	8	26	99
sc 56% bd	-	65	73	138
ec 50% bd	65	8	65	138
sc 80% ct	-	52	34	86
ec 80% ct	52	8	26	86
sc 90% ct	-	58	34	92
ec 90% ct	58	8	26	92
sc 100% ct	-	65	34	99
ec 100% ct	65	8	26	99

Tabela 2. Tamanhos de cache (em MB)

No primeiro conjunto de experimentos fixamos o tamanho do nível 1, a fim de que pudesse armazenar todos os termos de pesquisa da carga de trabalho, e variamos o tamanho dos níveis 2 e 3 para que armazenassem, localmente em cada servidor de transações, 10%, 20% e 50% dos livros contidos no banco de dados. Enquanto que no segundo conjunto de experimentos fixamos os níveis locais em 20% dos livros e variamos a capacidade do nível 1 para que absorvesse 80%, 90% e 100% da carga de trabalho.

Devido a implementação da eCache, que simplesmente adiciona um segmento global contendo um diretório de meta-dados, ainda precisamos do nível 1 local para obter os dados nos níveis 2 e 3. Por não termos nenhuma indicação de qual seria o tamanho adequado ao nível 1 local em cada nó de um servidor com eCache, adotamos um valor experimental de 8 MB, que permite armazenar aproximadamente 8.000 requisições de pesquisa. Por causa disso, aumentamos proporcionalmente o tamanho dos níveis 2 e 3 no servidor sem cooperação, a fim de mantermos o mesmo tamanho total de cache. Esta diferença de 6% em cada nó pode ser observada na Tabela 2.

Através destas configurações pretendemos avaliar o impacto da eCache no desempenho do servidor com o aumento do número de computadores que o compõem. No primeiro conjunto de testes avaliaremos a eCache em sua máxima cooperação com três graduações de replicação de dados. Com o segundo conjunto podemos medir o impacto do tamanho do segmento global no desempenho do servidor. Em todos os testes a cache não foi previamente aquecida, a fim de avaliarmos também seu comportamento sob a condição de "partida fria" (*cold-start*).

4.3. Carga de Trabalho

Para aplicar a carga de trabalho foi utilizado o software WebStone versão 2.0.1 [14], o qual simula um grupo de clientes enviando aleatoriamente as requisições ao servidor WWW por um determinado intervalo de tempo. Cada teste foi realizado com 16 clientes distribuídos em 4 computadores, com 3 execuções consecutivas de 30 minutos.

Tipo dao Operação	Nº de Requisições	Percentagem do Total
pay	172	0,08%
add	4.904	2,23%
browse	16.698	7,59%
home	20.765	9,43%
select	45.157	20,52%
search	132.399	60,16%

Tabela 3. Composição da carga de trabalho

Nos experimentos foram utilizadas somente requisições do tipo *search* retiradas de uma carga de trabalho com 220.095 requisições, obtida a partir de um *log* real de uma livraria virtual. A Tabela 3 apresenta as quantidades para cada tipo de requisição desta carga completa. Uma descrição mais detalhada sobre sua geração pode ser encontrada em [3, 13].

Removendo-se as informações sobre sessões de usuários e juntando-se as requisições idênticas, obtivemos um conjunto de 65.780 requisições de pesquisa. O número de vezes que cada uma delas aparece na carga original também

foi passado ao WebStone como um parâmetro de peso associado a requisição, ou seja, quanto maior este valor, maior a freqüência de envio.

Em 67% destas requisições o servidor retornou uma página de 1,4 KB informando que não encontrou nada. Os 33% restantes retornaram em média 214 livros por pesquisa, dos quais somente os primeiros 25 foram armazenados no nível 2 da cache e mostrados ao cliente. Nestes casos a página de resposta foi de 3,6 KB em média.

O tamanho médio dos termos de pesquisa nesta carga de trabalho é de 15 bytes. Com base nestes valores podemos afirmar que cada entrada no nível 1 da eCache, seja no segmento global ou local, gasta aproximadamente 1 KB para ser armazenada.

A decisão de testar o servidor somente com este conjunto de requisições de pesquisa se deve a dois fatores. Primeiro, as requisições do tipo *search* são as que mais fazem uso da eCache, e portanto, as mais relevantes para se testar o sistema. Segundo, são as de maior quantidade e de maior custo, ou seja, as que sobrecarregam mais o servidor.

4.4. Análise dos Resultados

Antes dos testes com as configurações de cache descritas acima, medimos a capacidade máxima de vazão de um único Apache servindo somente conteúdo estático, a fim de determinarmos o ponto de saturação do servidor. Respondendo sempre um mesmo arquivo de 3 KB ele atingiu a taxa máxima de 830 conexões por segundo. Por causa deste limite, os testes com a configuração `ec 50% bd` foram refeitos utilizando-se um computador extra como servidor WWW e dois conjuntos de 12 clientes, cada qual executando em 3 computadores e enviando requisições para somente um dos Apaches.

A Figura 3 mostra o tempo médio de resposta, em segundos, para cada configuração do primeiro conjunto de experimentos. Enquanto a Figura 4 mostra a vazão, em número de conexões por segundo, alcançada pelo servidor em cada caso. Destes resultados podemos observar claramente a redução no tempo de resposta com o uso da eCache e conseqüentemente um aumento significativo na capacidade do servidor de atender uma quantidade maior de requisições.

Estes experimentos também serviram para nos mostrar a boa escalabilidade do servidor até 8 nós no cenário em que a eCache consegue armazenar todas as requisições feitas em um certo intervalo de tempo. Todos os resultados foram acima de linear, exceto para a configuração mais restrita, `ec 10% bd`, com 8 nós, onde o servidor alcançou uma vazão somente 5,7 vezes maior em relação a 1 nó. Neste caso, o maior número de acessos ao banco de dados, devido ao tamanho restrito da cache e ao maior número de nós, não foi suficientemente compensado pela cooperação de cache.

Como esperado, o número de acertos (*hits*) na cache local aumentou nos dois tipos de cache à medida que se aumentou o tamanho da área de armazenamento local. Mas no caso do servidor sem cooperação este aumento não proporcionou uma melhora no desempenho. O servidor de banco de dados ainda se manteve como o gargalo do sistema, impondo muito custo às requisições. Já a eCache fez melhor proveito de um segmento local maior, pois além do ganho direto com mais *hits* locais, cada nó pôde encontrar na memória dos outros uma quantidade maior de livros que precisava para atender suas requisições.

Em média, uma requisição de pesquisa que acertou na cache local gastou 621 μs. Erros (*misses*) e acertos globais tiveram variações maiores conforme cada configuração. Comparando-os com o tempo de *hit* local, um *miss* foi em média 195 vezes mais lento e um *hit* global foi em média 41

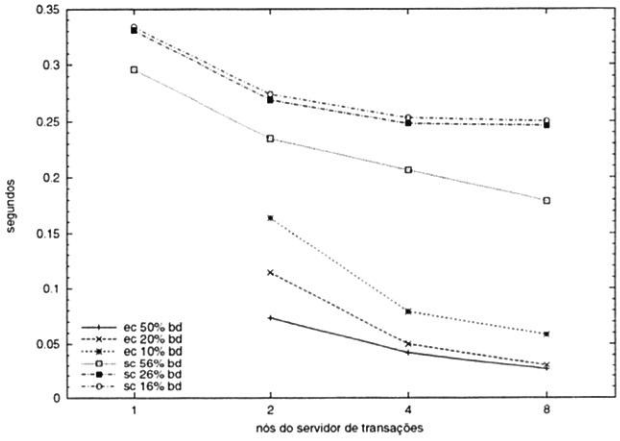

Figura 3. Tempo médio de resposta (variando cache local)

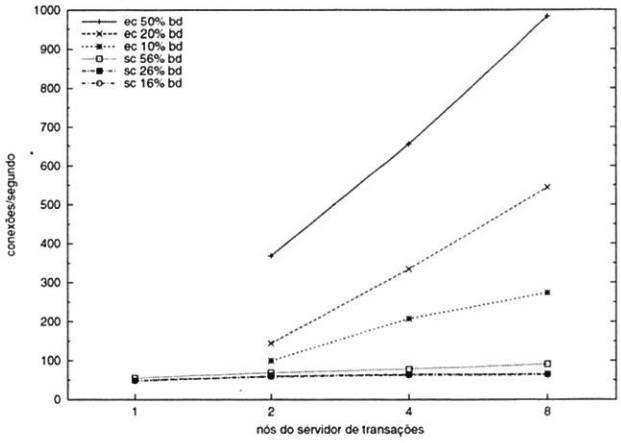

Figura 4. Taxa de conexões do servidor (variando cache local)

vezes mais lento. Embora os tempos de *hits* globais tenham sido uma ordem de grandeza maior que os de *hits* locais, eles foram em média 4,4 vezes mais rápidos do que um *miss*. Além disso, foram os responsáveis pela redução da carga sobre o banco de dados e conseqüentemente também reduziram o tempo médio de um *miss*.

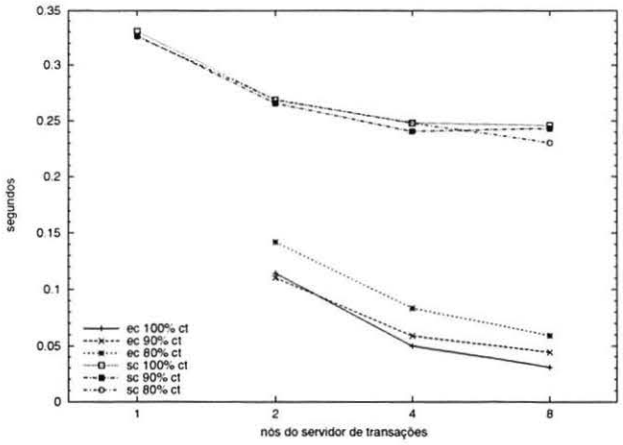

Figura 5. Tempo médio de resposta (variando cache global)

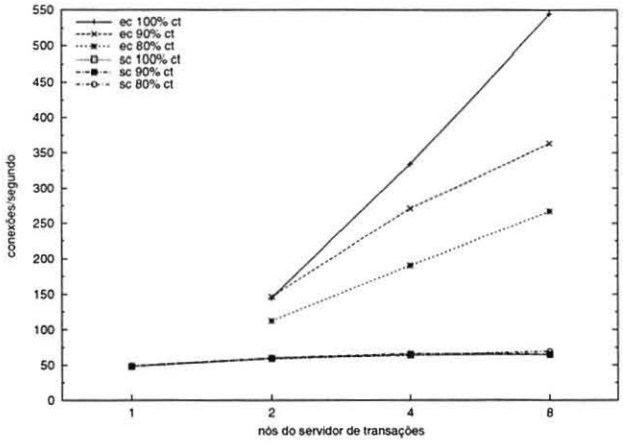

Figura 6. Taxa de conexões do servidor (variando cache global)

Em todos os testes pôde-se confirmar o aumento do tempo de *miss* à medida que se aumentou o número de nós do servidor de transações. Mesmo tendo sido considerável, utilizando eCache isto foi compensado pela redução do número de ocorrências de *misses*. Enquanto que no servidor sem cooperação, esta redução não ocorreu e acabou por comprometer ainda mais a escalabilidade.

Os resultados do segundo conjunto de experimentos, onde variamos o tamanho do primeiro nível da cache, são mostrados nas Figuras 5 e 6. Através deles pudemos confirmar e mensurar a já esperada degradação no desempenho do servidor. No cenário em que a eCache não consegue absorver as requisições de um certo intervalo de tempo, o servidor de banco de dados volta a ser um problema.

O pior caso constatado foi para a configuração `ec 80% ct` com 8 nós, onde o desempenho teve uma queda de 49%. A escalabilidade do servidor ficou comprometida à medida que a cooperação não foi suficiente para compensar o custo imposto pelo banco de dados. Mesmo assim, estes resultados se mantiveram melhores do que os sem cooperação.

Configuração	Nós	Tempo (s)	% do total
ec 50% bd	2	281	5,20
	4	358	6,63
	8	336	6,22
ec 20% bd	2	486	9,00
ec 100% ct	4	1568	29,04
	8	988	18,30
ec 10% bd	2	412	7,63
	4	1359	25,17
	8	1955	36,20
ec 90% ct	2	568	10,52
	4	1550	28,70
	8	1447	26,80
ec 80% ct	2	580	10,74
	4	1480	27,41
	8	1454	26,93

Tabela 4. Tempo total de execução de HLRC

O tempo gasto com HLRC foi considerável em alguns dos testes, como pode-se ver na Tabela 4. O valores mostram o tempo total, em segundos e percentual, de HLRC durante cada teste completo de 90 minutos. Mas este fato era totalmente esperado, já que o servidor com eCache faz uso intenso das informações armazenadas na área de memória compartilhada. O real custo imposto pelo software DSM, quando comparado a uma implementação somente com passagem de mensagens, não foi avaliado nestes experimentos, sendo deixado para trabalho futuro.

5. Trabalhos Relacionados

Cache cooperativa [7] tem sido usada como uma boa solução para reduzir a sobrecarga dos servidores, permitindo se obter dados que estejam na cache de outros clientes.

Sarkar e Hartman [16] apresentam uma cache cooperativa para sistemas de arquivos distribuídos usando um algoritmo baseado em dicas (*hints*). Eles mostraram que trabalhar com informação global imprecisa funciona tão bem quanto um estado global preciso do sistema e também oferece um custo de execução mais baixo. Cada cliente do

sistema de arquivos mantém um estado aproximado para os blocos de acesso presentes na cache cooperativa, sem consultar um gerente centralizado.

Uma idéia semelhante foi usada no Cooperative Caching Middleware (CCM) [6] que é uma camada genérica de cache cooperativa baseada em blocos. Neste trabalho eles compararam, através de simulação, o desempenho de dois servidores Web, um usando CCM e outro usando uma distribuição otimizada que verifica a carga dos nós no cluster e o conteúdo das requisições para aproveitar melhor a cache local de cada um. A vantagem está na generalidade do CCM, que permite a construção de diversos serviços distribuídos, reduzindo os esforços para projetar e implementar servidores baseados em cluster. Sua desvantagem pode ser o desempenho um pouco inferior. CCM foi avaliado em um servidor Web com conteúdo estático.

Nossa implementação difere destes trabalhos por usar software DSM para manter um estado global mais exato do que *hints*, e por não migrar entradas de cache para outros nós em sua política de substituição, como faz CCM.

Swala [10], assim como em nossa versão preliminar da eCache [9], também usa cache cooperativa para conteúdo dinâmico em um servidor Web distribuído. A diferença é que Swala armazena o resultado de requisições a programas CGI em um módulo do servidor Web. Vários desses servidores cooperam trocando meta-dados sobre suas caches, que são mantidos em memória, mas os resultados são armazenados em arquivos. Assim, mesmo uma consulta bem sucedida à cache acaba gerando uma busca no sistema de arquivos.

6. Conclusões

Nós apresentamos e avaliamos a eCache, uma cache cooperativa no nível da aplicação de comércio eletrônico para servidores baseados em cluster. A cooperação entre as caches locais melhora a vazão do servidor à medida que alivia a demanda sobre seu banco de dados. Nossa implementação utiliza duas abordagens de programação: passagem de mensagens para trocar dados entre os nós do cluster e software DSM para manter a informação global de qual nó da aplicação possui um determinado dado em sua cache local.

Nossos experimentos mostram que a eCache pode melhorar o desempenho do servidor, reduzindo significativamente o número de consultas ao banco de dados. Também pudemos verificar a boa escalabilidade do servidor fornecida pela cooperação de cache. No caso do servidor com 8 nós usando eCache e conseguindo armazenar boa parte das requisições de pesquisa, atingimos uma taxa de conexões 10 vezes melhor do que o servidor sem este recurso. Mesmo na configuração de cache mais restrita, nossa implementação funcionou bem, aumentando mais de 4 vezes a vazão do servidor.

Referências

[1] D. Andresen, T. Yang, V. Holmedahl, and O. Ibarra. SWEB: Towards a Scalable World Wide Web Server on Multicomputers. In *Proceedings of the 10th IEEE IPPS*, pp. 850–856, Honolulu, HI, April 1996.

[2] M. Aron, D. Sanders, P. Druschel, and W. Zwaenepoel. Scalable Content-ware Request Distribution in Cluster-based Network Servers. In *Proceedings of the USENIX Annual Technical Conference*, San Diego, CA, June 2000.

[3] T. Cançado, A. Pereira, B. Abrahão, et al. Servidores Paralelos e Distribuídos de Comércio Eletrônico. In *Anais do II Workshop em Sistemas Computacionais de Alto Desempenho*, pp. 103–110, Pirenópolis, GO, Setembro 2001.

[4] E. Carrera and R. Bianchini. Evaluating Cluster-Based Network Servers. In *Proceedings of the 9th IEEE International Symposium on High Performance Distributed Computing*, pp. 63–70, August 2000.

[5] Compaq Computer Corp., Intel Corp., and Microsoft Corp. *Virtual Interface Architecture Specification, Version 1.0*, December 1997. `http://www.intel.com/design/servers/vi/the_spec/specification.htm`.

[6] F. M. Cuenca-Acuna and T. D. Nguyen. Cooperative Caching Middleware for Cluster-Based Servers. In *Proceedings of Tenth IEEE International Symposium on High Performance Distributed Computing*, August 2001.

[7] M. Dahlin, R. Wang, T. Anderson, and D. Patterson. Cooperative Caching: Using Remote Cliente Memory to Improve File System Performance. In *Proc. of the First Symposium on Operating Systems Design and Implementation*, 1994.

[8] D. Dias, W. Kish, R. Mukherjee, and R. Tewari. A Scalable and Highly Available Web Server. In *Proceedings of the IEEE COMPCON*, pp. 85–92, San Jose, CA, February 1996.

[9] S. Dias, A. Silva, et al. eCache: a User-level Cooperative Cache for e-Commerce Cluster-based Network Servers. Technical Report ES-587/02, Universidade Federal do Rio de Janeiro, Dezembro 2002.

[10] V. Holmedahl, B. Smith, and T. Yang. Cooperative Caching of Dynamic Content on a Distributed Web Server. In *Proc. of Seventh IEEE International Symposium on High Performance Distributed Computing*, pp. 243–250, July 1998.

[11] L. Iftode. *Home-Based Shared Virtual Memory*. PhD thesis, Princeton University, June 1998.

[12] W. Meira Jr., C. Murta, S. Campos, and D. Guedes. *Comércio Eletrônico: Projeto e Desenvolvimento de Sistemas*. Edições Campus-SBC, Março 2002.

[13] D. Menascé, V. Almeida, et al. In Search of Invariants for E-Business Workloads. In *Proc. of the 2nd ACM e-Commerce Conference*, pp. 56–65, Minneapolis, MN, October 2000.

[14] Mindcraft, Inc. WebStone. `http://www.mindcraft.com/webstone/ws201index.html`.

[15] V. Pai, M. Aron, G. Banga, et al. Locality-Aware Request Distribution in Cluster-based Network Servers. In *Proceedings of the 8th ASPLOS*, San Jose, CA, October 1998.

[16] P. Sarkar and J. Hartman. Efficient Cooperative Caching using Hints. In *Proceedings of the 2nd Symposium on Operating Systems Design and Implementation*, 1996.