CACHEAMENTO TRANSPARENTE DE BANCO DE DADOS EM SERVIDORES DE COMÉRCIO ELETRÔNICO

Antônio Marcelo Azevedo Alexandre

DISSERTAÇÃO SUBMETIDA AO CORPO DOCENTE DA COORDENAÇÃO DOS PROGRAMAS DE PÓS-GRADUAÇÃO DE ENGENHARIA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

Aprovada por:

____
Prof. Cláudio Luis de Amorim, Ph.D.

____
Prof . Marta Lima Queirós Mattoso, D.Sc.

____
Prof. Wagner Meira Junior, Ph.D.

RIO DE JANEIRO, RJ – BRASIL
AGOSTO DE 2005

ALEXANDRE, ANTÔNIO MARCELO AZEVEDO

Cacheamento Transparente de Banco de Dados em Servidores de Comércio Eletrônico [Rio de Janeiro] 2005

XII, 112 p. 29,7 cm (COPPE/UFRJ, M.Sc., Engenharia de Sistemas e Computação, 2005)

Dissertação – Universidade Federal do Rio de Janeiro, COPPE

1. Comércio Eletrônico
2. Cache Consistente para Bancos de Dados
3. Java

I. COPPE/UFRJ II.Título (série)

A todos aqueles que também acreditam que ciência e tecnologia só fazem sentido se para um mundo melhor e mais justo para todos.

Agradecimentos

Aos meus pais, Jorge e Carmen, que com amor, devoção e sacrifícios nunca medidos, possibilitaram mais esta etapa da minha educação. E também pela formação moral e ética que procuraram me oferecer, tão útil nesta atividade que nada mais é do que a busca da verdade.

À minha mulher, Yoshie, amiga e companheira, que sempre soube com muita ternura compreender aquilo que resultava das minhas tentativas de superação dos desafios: o inevitável *stress*. E que o fazia desaparecer como que por mágica apenas com seu sorriso encantador e algumas poucas palavras.

Ao meu querido irmão Flávio, por não ter se importado quando por falta de tempo eu tinha que me furtar de nossos agradáveis e criativos bate-papos técnicos.

Aos parentes que acolheram a mim e à minha mulher na cidade do Rio de Janeiro, muito numerosos para que eu os cite individualmente, e que, cada um à sua forma, ajudaram-nos a nos sentir um pouco menos estrangeiros nesta cidade. Meus agradecimentos especiais ao tio Fernando Alexandre, que nos hospedou em nossos primeiros dias no Rio, e ao primo Miguel Pereira, que, por experiência própria consciente como ninguém da importância da educação e da formação acadêmica, nos incentivou de todas as maneiras possíveis.

Aos colegas da minha turma de mestrado, dentre os quais, Roberto Ligeiro, Fernando Miranda, Carlos Braga e Isaac Nogueira, com quem passei muitas horas em grupos de estudo no início do curso, e que provaram que, mesmo em um ambiente que parece incentivar e premiar a individualidade e a competição, é possível também a cooperação e a solidariedade nos momentos em que isso é cabível.

Aos colegas do Laboratório de Computação Paralela, especialmente Marcelo Lobosco, Leonardo Pinho e Lauro Whately, que sempre mostraram boa vontade no auxílio de uso e manutenção da nossa excelente infra-estrutura comum do laboratório. Ao também colega Sérgio Kostin e sua esposa Simone Vendramini, competente fisioterapeuta, pelo auxílio com um problema ortopédico que chegou a ameaçar seriamente minha produtividade. A todos os demais pelas críticas e sugestões.

Ao meu orientador Claudio Amorim, por ter apostado na minha proposta de trabalho, aceitado-me em seu grupo e laboratório, e por ter me mostrado caminhos que não eram evidentes para mim. Obrigado pelo aprendizado proporcionado.

Resumo da Dissertação apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências (M.Sc.)

CACHEAMENTO TRANSPARENTE DE BANCO DE DADOS EM SERVIDORES DE COMÉRCIO ELETRÔNICO

Antônio Marcelo Azevedo Alexandre

Agosto/2005

Orientador: Cláudio Luis de Amorim

Programa: Engenharia de Sistemas e Computação

A popularização do uso da Internet transformou a rede em um meio em muitos casos preferencial à prestação de serviços. Entretanto, para enfrentar essa demanda crescente, estão sujeitos a cargas cada vez maiores os servidores web, de aplicação e principalmente os de bancos de dados, que acabam se tornando cada vez mais um gargalo em sistemas para web. O cacheamento de banco de dados pode ajudar a aliviar esse gargalo, mas as alternativas não transparentes de cacheamento de banco de dados exigem que os programadores de aplicações modifiquem o código fonte de seus sistemas para se valerem do cacheamento, e que tenham cuidado especial em não causar problemas de coerência de cache. Um sistema de cacheamento foi projetado e implementado como um driver JDBC (Java Database Connectivity) para oferecer suporte a cacheamento transparente de banco de dados com garantia de consistência a aplicações de comércio eletrônico baseadas na plataforma Java. Os resultados experimentais, obtidos para um agregado de máquinas composto de um servidor web, um servidor de aplicação implementando a lógica de negócio do sistema de comércio eletrônico e de um servidor de banco de dados, mostraram ganhos de desempenho quando a CPU da máquina servidora de banco de dados era o gargalo.

Abstract of Dissertation presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Master of Science (M.Sc.)

TRANSPARENT DATABASE CACHING IN E-COMMERCE SERVERS

Antônio Marcelo Azevedo Alexandre

August/2005

Advisor: Cláudio Luis de Amorim

Department: Computing and Systems Engineering

The wide acceptance of the Internet among the general public has turned the net into a preferential means of providing commercial and general interest services to the public. Nevertheless, in order to cope with the resulting demand of such tendency, increasing load is experienced by web, application, and specially database servers, which become more and more a major bottleneck in web-based systems. Database caching may help relieve such bottleneck, but non-transparent database caching alternatives require that application programmers modify their source code to benefit from database caching, and thus be specially mindful of potential cache coherence problems. A database caching system was designed and implemented as a JDBC (Java Database Connectivity) driver to provide transparent consistent cache support to e-commerce applications based on the Java platform. Experimental results for a cluster of machines consisting of a front-end web server, an application server implementing the e-commerce business logic, and a back-end database server, have shown performance gains when the CPU of the database server machine was the bottleneck.

Sumário

1 Introdução 1

2 Servidores de comércio eletrônico 7

2.1 *Clusters* para *Web* 7

2.1.1 Diminuindo o gargalo de acesso ao banco de dados. 9

2.2 Servidores de comércio eletrônico 10

2.2.1 O *Benchmark* TPC-W 14

2.2.1.1 Cálculo da vazão no TPC-W 16

2.3 *Drivers* JDBC 20

2.3.1 Consultas preparadas parametrizadas 21

2.3.2 Isolamento de transações 23

2.3.3 Acesso a metadados 25

2.3.4 Tipos de *drivers* JDBC 26

2.3.5 Gerente de *Drivers* 29

3 A arquitetura do dCache 30

3.1 Arquitetura do dCache 31

3.1.1 Estruturas de cache 32

3.2 Funcionamento básico das consultas 33

3.3 Atualizações e invalidações 35

3.3.1 Atualizações 36

3.3.2 Invalidações 40

3.3.3 Suspensão temporária do uso da cache 42

3.4 Limitações do *driver* 45

4 Avaliação experimental do dCache 48

4.1 Ambiente experimental 49

4.2 Benchmark utilizado 50

4.3 Descrição dos experimentos 52

4.3.1 Instrumentação do *driver* 53
4.4 Primeira série 54
4.4.1 Desempenho 54
4.4.2 Uso de memória 61
4.5 Segunda série 65
4.5.1 Desempenho 67
4.6 Discussão 70

5 Trabalhos relacionados **72**

6 Conclusões **76**

A Gramática do analisador sintático **79**
A.1 Regras de produção 79
A.2 Tokens 84

B Interface gráfica **86**

C Dados experimentais em detalhe **89**

Lista de Figuras

2.1 Cluster Web 8
2.2 Customer Behavior Model Graph 12
2.3 Schema do banco de dados do TPC-W 15
2.4 Interações do TPC-W 17
2.5 Três passos básicos que um desenvolvedor de aplicação realiza para acessar dados por meio de JDBC 21
2.6 Consultas preparadas parametrizadas 22
2.7 Atualizações e transações 24
2.8 Tipos de *driver* JDBC 28

3.1 Arquitetura do dCache 31
3.2 Estruturas internas de cache do *driver* dCache 32
3.3 Funcionamento da cache de comandos SQL e da cache de parâmetros para consultas. 33
3.4 Relação entre cache de parâmetros e cache de tuplas 34
3.5 Relação entre cache de comandos SQL e cache de tuplas 34
3.6 Exemplo de construção do vetor de dependência da projeção. . . . 41
3.7 Falhas por invalidação 43
3.8 Suspensão temporária do uso da cache 44

4.1 Arquitetura do ambiente de hardware utilizado nos experimentos . . 50
4.2 Interações Web por minuto, com e sem cache, usando-se as cargas Browsing (WIPMb), Shopping (WIPM) e OLTP (WIPMo) - primeira série 55
4.3 Taxa de ocupação da CPU das máquinas com o servidor de banco de dados (a), servidor de aplicação (b) e servidor Web (c) - primeira série 56
4.4 Tempo aguardando banco de dados, em segundos, por navegador, (a) sem cache e (b) com cache - primeira série 57

4.5 Taxa de acertos (a) e taxa de falhas por invalidação e por suspensão do uso da cache (b) - primeira série 58
4.6 Custo de falhas fracionado entre interações, em microssegundos (a) e fracionamento da ocorrência das interações - primeira série 60
4.7 Ocupação das caches de tuplas para as três cargas (a), na primeira série de experimentos e as proporções de ocorrência de cada interação. 63
4.8 Distribuição acumulada da probabilidade de ocorrência das strings de procura por título (a) e autor (b), e da ocorrência das ofertas (c), conforme a posição no ranking, em 100.000.000 de requisições, para um banco de dados de 100k itens. 66
4.9 Interações Web por minuto, com e sem cache, usando-se as cargas Browsing (WIPMb), Shopping (WIPM) e OLTP (WIPMo) - segunda série 68
4.10 Taxa de ocupação da CPU das máquinas com o servidor de banco de dados (a), servidor de aplicação (b) e servidor Web (c) - segunda série 68
4.11 Custo de falhas fracionado entre interações, em microssegundos (a) e fracionamento da ocorrência das interações (b) - segunda série . . 69

6.1 Arquitetura de cluster com o dCache e vários servidores de aplicação 78

B.1 Schema do banco de dados em detalhe 87
B.2 À esquerda, a árvore sintática de uma constulta SQL em detalhe, à direita a cache de parâmetros em detalhe. 88

C.1 Comandos S1 a S13 91
C.2 Comandos S14 a S24 92
C.3 Comandos S25 e S26 93
C.4 Comandos I1 a I5 e U1 a U3 94

Lista de Tabelas

2.1 Quantidade média de visitas a cada estado para as sessões dos grafos A e B da figura 2.2. 13
2.2 Proporção das interações com operações de leitura, leitura e escrita, e que não envolvem o uso do banco de dados, conforme a carga. . . 14
2.3 Proporções das interações do TPC-W conforme os três tipos de carga. 18

4.1 Tempo aguardando banco de dados, em segundos, por navegador - primeira série 58
4.2 Taxa de acertos, invalidações e taxa de falhas por invalidação - consultas S6 e S15 61
4.3 Entradas da cache de tuplas, de parâmetros e das tabelas de dependência global usadas na primeira série de experimentos 62
4.4 Ocupação máxima das caches de tuplas, em número de entradas, para as três cargas de acordo com o tipo de consulta, e a quantidade de tuplas retornadas a cada consulta - primeira série. 64
4.5 Efeito das falhas por invalidação no número de entradas das caches de tuplas de três tipos de consulta - OLTP - primeira série 65
4.6 Redução do tempo aguardando banco de dados devido ao uso da cache 70

C.1 Tamanho das tabelas utilizadas na série 1 (100k itens, 300 nav.) e série 2 (1M itens, 50 nav.) de experimentos 89
C.2 Abreviaturas usadas para as interações Web 90
C.3 Consultas e as interações web em que ocorrem 95
C.4 Consultas do Browsing Mix, com uso de cache - primeira série (100k itens, 300 nav.) 96
C.5 Consultas do Shopping Mix, com uso de cache - primeira série (100k itens, 300 nav.) 97
C.6 Consultas do Ordering Mix, com uso de cache - primeira série (100k itens, 300 nav.) 98

C.7 Operações de escrita no banco de dados, Browsing Mix - primeira série (100k itens, 300 nav.) 99
C.8 Operações de escrita no banco de dados, Shopping Mix - primeira série (100k itens, 300 nav.) 100
C.9 Operações de escrita no banco de dados, Ordering Mix - primeira série (100k itens, 300 nav.) 101
C.10 Consultas do Browsing Mix, com uso de cache - segunda série (1M itens, 50 nav.) 102
C.11 Consultas do Shopping Mix, com uso de cache - segunda série (1M itens, 50 nav.) 103
C.12 Consultas do Ordering Mix, com uso de cache - segunda série (1M itens, 50 nav.) 104
C.13 Operações de escrita no banco de dados, Browsing Mix - segunda série (1M itens, 50 nav.) 105
C.14 Operações de escrita no banco de dados, Shopping Mix - segunda série (1M itens, 50 nav.) 106
C.15 Operações de escrita no banco de dados, Ordering Mix - segunda série (1M itens, 50 nav.) 107

Capítulo 1

Introdução

A popularização do acesso à Internet transformou a rede em um meio em muitos casos preferencial à prestação de serviços a cidadãos e a consumidores. Os usuários dos serviços WWW (World-Wide Web) já são milhões, o que impõe a busca tecnológica de infraestrutura capaz de acompanhar essa crescente demanda. Dois dos maiores obstáculos tecnológicos que a infraestrutura na Internet encontra são, por um lado, a velocidade e latência das redes de transmissão de dados, e, por outro, a capacidade de processamento dos serviços Web.

A insuficiência desses recursos traduz-se, para o usuário da Internet, na desconfortável demora, ou latência, na apresentação das páginas pelo seu navegador. Essa deficiência pode determinar a insatisfação ou mesmo desistência por parte do usuário em usar o serviço, o que é inaceitável quando o serviço em questão é oferecido exclusivamente pela Internet. Esse é o caso de muitos negócios baseados em comércio eletrônico, sem existência física fora da rede, e que por isso dependem da suficiência desses recursos técnicos de infraestrutura.

Ainda que a largura de banda nas redes seja um grande problema nesse contexto, é previsto que a carga nos servidores Web se torne cada vez mais o grande gargalo no futuro [1]. A tentativa de expandir sistemas baseados em uma única máquina servidora não é uma solução economicamente viável a longo prazo, especialmente em vista do crescimento costumeiro na demanda por parte dos usuários na Internet, muitas vezes abrupto. Uma alternativa viável e já popular é a construção de servidores *Web* baseados em aglomerados de computadores, ou *clusters*.

Os *clusters* de servidores *Web* servem não apenas páginas estáticas, mas também conteúdo dinâmico, como é o caso de grande parte dos serviços na Internet que têm um nível mínimo de complexidade. Esses *clusters* são compostos por máquinas com software de serviço *Web* [1], para servir as páginas estáticas, e por uma associação entre sistemas gerenciadores de bancos de dados (SGBD) e servidores de aplicação

- também chamados de servidores de transação - que servem as páginas dinâmicas.

Mas o fluxo de dados resultante da comunicação entre os nós de um *cluster Web* pode estar também sujeito a gargalos, o que compromete o tempo de resposta do serviço. O acesso ao SGBD é um importante gargalo em sistemas de conteúdo dinâmico para a *Web* [2, 3]. Ainda que muitas operações internas a um SGBD possam ser feitas em paralelo, manter o compromisso entre atomicidade, coerência e isolamento das transações em paralelo e ainda assim manter o desempenho alto e satisfatório é uma tarefa de implementação complexa, especialmente considerando-se que a comunicação em *clusters* é feita com troca de mensagens entre os nós, via rede. Em vista disso, servidores paralelos para bancos de dados podem ser uma modalidade de software muito caro, o que limita o seu uso a grandes empresas e organizações[1].

A utilização da técnica de *cache* pode ajudar a reduzir a contenção e o tempo de acesso ao SGBD em um *cluster* para *Web*, uma vez que não é necessário o envolvimento do servidor de banco de dados em consultas anteriormente realizadas, e cujos dados buscados ainda estejam na memória *cache*. Do ponto de vista do desenvolvimento da aplicação, uma *cache* para aplicações que fazem acesso a bancos de dados pode ser implementada das seguintes maneiras:

1. *Implementando-se a funcionalidade da cache especificamente para uma aplicação.* Essa abordagem foi usada em um sistema chamado *eCache* [9], que em parte motivou o desenvolvimento deste trabalho. O *eCache* é um sistema servidor de comércio eletrônico destinado a demonstrar o uso de duas técnicas de programação, memória compartilhada distribuída e passagem de mensagens, na construção de uma loja virtual com uma cache cooperativa entre nós de um *cluster*. Nesse sistema, a carga imposta ao banco de dados é substancialmente reduzida com o uso de uma *cache* global, única a todo o *cluster* e mantida em memória compartilhada, e de várias *caches* locais, pertencentes a cada nó do *cluster*, e usando segmentos de memória locais. A cache global contém um diretório distribuído de meta-dados, usado para identificar a localização dos dados que ficam armazenados na cache local de cada nó do *cluster*. Por ter sido feita especificamente para a aplicação em questão - uma loja virtual de livros - a implementação dessas caches não tem generalidade suficiente para ser usada diretamente em outras aplicações.

[1] Considera-se o lançamento, em meados de 2004, do "MySQL Cluster" [4], SGBD paralelo de código aberto e livre, e com modelo de replicação síncrona, um fator que pode ajudar a reverter essa tendência no futuro. Outras implementações de SGBD com replicação síncrona [5, 6], assíncrona [7] e de processamento paralelo [8, 13] baseadas em *software* livre têm surgido nos últimos anos.

2. *Utilizando-se "frameworks" ou bibliotecas de cache genéricas.* Várias opções existem para o uso de bibliotecas genéricas que implementam *caches* [10, 11, 27]. Entretanto, é necessária uma referência explícita a algum identificador no código-fonte cada vez em que se cria ou obtém um novo objeto que deve estar presente em *cache.* Isso significa que uma aplicação que não foi implementada inicialmente para usar uma *cache* precisa sofrer várias modificações para que ela possa ser usada. De modo semelhante, a troca de uma biblioteca de *cache* por uma outra que ofereça recursos mais vantajosos envolve obrigatoriamente alterações ao código-fonte. Essas modificações podem se tornar uma fonte a mais de erros, além de misturarem indesejavelmente no código-fonte aspectos de desempenho do sistema com a lógica de negócio da aplicação.

3. *Embutindo-se a funcionalidade de cache em algum "driver" ou em algum "middleware" que realiza o acesso ao SGBD.* Em diversas plataformas, o acesso ao servidor de banco de dados é realizado unicamente e de forma padronizada por meio de interfaces de programação de aplicações (API). Essas interfaces visam a liberar o programador da tarefa de conhecer o protocolo de rede usado para comunicação com o servidor de banco de dados, além de fornecer um meio de garantir a portabilidade da aplicação caso o SGBD seja trocado por um de outro fabricante. Do ponto de vista de programação, o *driver* que implementa essa API, por ser um ponto central no acesso ao banco de dados, é um ponto interessante para se embutir a implementação de uma cache. Da mesma forma, um *middleware* que se coloca entre a aplicação e o servidor de bancos de dados também é um ponto central de acesso, e portanto também é um bom candidato a possuir uma *cache* embutida. Alguns exemplos de implementação dessa estratégia mostram que as aplicações podem se valer dos benefícios da *cache* sem que seja necessário alterá-las [13, 14, 15, 16], ou seja, mantendo intacto e sem alterações o código-fonte de uma aplicação que não utilizava cache originalmente.

Para essas caches embutidas, pelo menos dois modelos de armazenamento são possíveis:

1. *Cache de consultas*, em que se armazena apenas o resultado associado a uma consulta ao banco de dados. A consulta indexa diretamente o resultado na cache, e portanto o *hit* (acerto) em *cache* ocorre quando uma consulta é repetida exatamente como foi indexada.

2. *Cache de tabelas*, em que uma consulta é decomposta de forma que os dados são buscados das tabelas originais no banco, e armazenadas novamente em

tabelas, localmente em *cache*, como um subconjunto das tabelas do banco de dados. Quando uma nova consulta é feita, os dados são procurados nas tabelas locais da *cache*, e os acertos ocorrem individualmente em cada tabela.

No sistema eCache, cada acesso ao banco de dados em um nó do *cluster* é precedido por uma verificação da existência do dado em questão na *cache* local daquele nó. Caso o dado exista na *cache* local, ele é diretamente aproveitado pela aplicação, poupando-se o SGBD. Caso não se encontre o dado, é consultada a *cache* global, onde, em caso de sucesso, será encontrado o meta-dado que informa em quais nós do *cluster*, ou em quais *caches* locais, o dado pode ser encontrado. Finalmente e só então, caso o dado não exista em nenhuma cache local, é feita uma consulta ao banco de dados. Visto que um dado que está na *cache* local de um nó pode ser aproveitado pelo processo da aplicação que está em execução em outro nó, o sistema é dito de *cache cooperativa* [17, 18].

Ainda que o sistema demonstre a viabilidade e vantagens de uma *cache* cooperativa, dois problemas não foram abordados no eCache:

1. Todos os acessos ao SGDB são para leitura apenas. Não são realizadas escritas (atualizações) ao banco de dados, tais como inserções de novos dados, remoções ou alterações de valores. Não existe portanto no sistema a preocupação com a consistência dos dados na *cache* diante de atualizações ao banco de dados. Ainda que de maneira geral a quantidade de leituras em serviços oferecidos pela Internet seja na prática maior do que a quantidade de escritas[2], as escritas existem e precisam ser consideradas, sob pena de produzirem resultados incorretos e inadmissíveis. Um exemplo prático seria uma modificação de preço de um produto, feita por um responsável da loja virtual, não se manifestar nos valores antigos que constam na *cache*. Com isso, quando houvesse uma consulta posterior, o preço antigo seria mostrado ao usuário. São necessárias portanto invalidações ou atualizações dos valores em *cache*.

2. A *cache* implementada no eCache, conforme descrito anteriormente, foi desenvolvida especificamente para a aplicação. Uma implementação mais genérica seria desejável, porque facilitaria a avaliação da *cache* em outras aplicações. Ainda mais vantajoso seria implementar a *cache* embutida em um *driver* ou em um *middleware*, poupando-se o esforço de alterar cada aplicação que se pretenda que tire benefícios da *cache*, e colocando o conceito mais aplicável

[2]Observe-se que nem todos os usuários enviam informações com freqüência pelos seus navegadores, mas todos as recebem, e que o padrão de acesso mais comum na navegação é receber muito mais dados do que enviar dados aos servidores.

na prática à realidade de desenvolvimento de software comercial. A tendência atual em desenvolvimento de sistemas comerciais complexos é que os programadores sejam poupados de aspectos funcionais da infraestrutura de software e hardware, e concentrem-se apenas na lógica de negócio das aplicações. Essa tendência é refletida por exemplo nos componentes "Enterprise Java Beans (EJB)"[19] presentes na arquitetura de desenvolvimento J2EE [20], da Sun Microsystems.

Assim, este trabalho tem como objetivo a avaliação de uma *cache* de resultados de consultas SQL a bancos de dados implementada em um *driver*, para aplicações que tenham o padrão de acesso típico de serviços *Web* para a Internet, ou seja, um número de consultas ao banco de dados consideravelmente maior que o número de atualizações. A cache é genérica o suficiente para permitir que qualquer aplicação dessa natureza possa se aproveitar dela sem que tenham que ser feitas modificações em seu código-fonte. Essa cache foi batizada de "dCache", e a implementação é a de um *driver* JDBC, o que a qualifica para ser usada em aplicações escritas na linguagem Java.

O problema central no projeto e implementação de uma *cache* com esses requisitos é como manter a consistência dos dados em *cache* de forma transparente, visto que as aplicações que fazem uso dela não manipulam explicitamente os dados nela residentes. A rigor, a existência da cache nem mesmo é assumida pela aplicação. Consultas e atualizações são feitas por meio da linguagem SQL (Structured Query Language). Uma operação de escrita, feita a partir de uma requisição em uma linha de execução, precisa de maneira automática, a partir dos comandos em SQL, invalidar ou atualizar valores em *cache* acessíveis por outras requisições posteriores, em linhas de execução do mesmo ou de outro nó do *cluster*.

O uso do driver implementado mostrou resultados importantes ao explorar, com a cache em um servidor de aplicação, a redundância de consultas a banco de dados, que antes dependiam exclusivamente do SGDB. Entretanto, a presente implementação do driver limita-se ao funcionamento junto a um servidor de aplicação composto de um único nó. A extensão do driver para utilização de uma cache cooperativa, ou mesmo de uma cache que mantenha consistência entre vários nós, será um trabalho futuro. Os resultados apontam, porém, a possibilidade de explorar uma implementação paralela porque o SGBD em muitos casos mostrou-se um gargalo, e o driver provou reduzir substancialmente o uso de CPU do servidor de banco de dados, mesmo para a configuração com um único servidor de aplicação.

A descrição do mecanismo interno desse *driver* é feita no capítulo 3. A descrição de como funcionam os *drivers* JDBC em geral, é feita no capítulo 2. Visto que

a implementação é destinada a avaliar o uso de *cache* para aplicações *Web* de uso intenso, e que o padrão de acesso ao banco de dados assumido é comum em aplicações de comércio eletrônico, *clusters* para *Web* para aplicações de comércio eletrônico são assuntos abordados no capítulo 2. Também neste capítulo, é descrito o *benchmark* TPC-W, que modela uma loja virtual de livros, e tem o objetivo de testar a capacidade e a relação entre preço e desempenho de *hardware* e *software* para esse tipo de aplicação. Esse *benchmark* foi usado para avaliar o dCache.

No capítulo 4, é feita a avaliação experimental do dCache, que para uma configuração de *cluster* para Web composta de um servidor Web, um servidor de aplicação e um servidor de bancos de dados, demonstrou ganhos em termos do tempo de resposta e de interações por minuto para o *benchmark* TPC-W, assim como em redução do uso do SGBD. Esses ganhos foram maiores quanto maior era a taxa de ocupação do SGBD, em virtude do uso de consultas complexas ou envolvendo muitos registros, e quanto maior era a razão entre operações de leitura e operações de escrita no banco de dados.

Trabalhos relacionados a este são relatados no capítulo 5. E finalmente, as conclusões desta dissertação são tratadas no capítulo 6.

Capítulo 2

Servidores de comércio eletrônico

Boa parte da demora percebida por clientes da *Web* quando acessam serviços da *Web* é devida à carga nos servidores. A banda de rede é também um problema, mas os prognóticos apontam que a carga nos servidores vai se tornar crescentemente o grande gargalo no futuro [1]. Expandir servidores utilizando a capacidade de uma única máquina servidora não é uma solução econômica a longo prazo, especialmente em vista da tendência de crescimento da demanda sobre os servidores *Web*.

A adoção de clusters para web é a solução economicamente viável para a demanda sobre servidores de comércio eletrônico. Por isso eles são um assunto deste capítulo. Também são tratados neste capítulo os servidores de comércio eletrônico, do ponto de vista da caracterização de carga, e incluindo o benchmark TPC-W. Por último, é feita a descrição funcional dos *drivers* JDBC, que realizam a comunicação entre aplicações e SGDBs, e são um componente essencial em servidores de comércio eletrônico com servidores de aplicação baseados em Java.

2.1 *Clusters* para *Web*

Um *cluster* para *Web* é uma coleção de máquinas interconectadas por uma rede local de alta velocidade e fisicamente colocadas em um mesmo lugar, com o propósito de apresentar aos clientes da *Web* a imagem de um único servidor. Os *clusters* para *Web* são uma solução viável para o crescimento explosivo no tráfego da *Web* que vem ocorrendo na Internet nos últimos anos.

A figura 2.1 mostra uma arquitetura típica para clusters Web. Os clientes fazem os pedidos de páginas usando o protocolo HTTP. Um comutador (*switch*) Web pertencente ao cluster tem o endereço IP externo para o cluster, e encaminha os pedidos para os servidores Web reais no cluster. Também chamado de *frontend*, ele é um despachante centralizado, e visto que ele tem um controle de granularidade

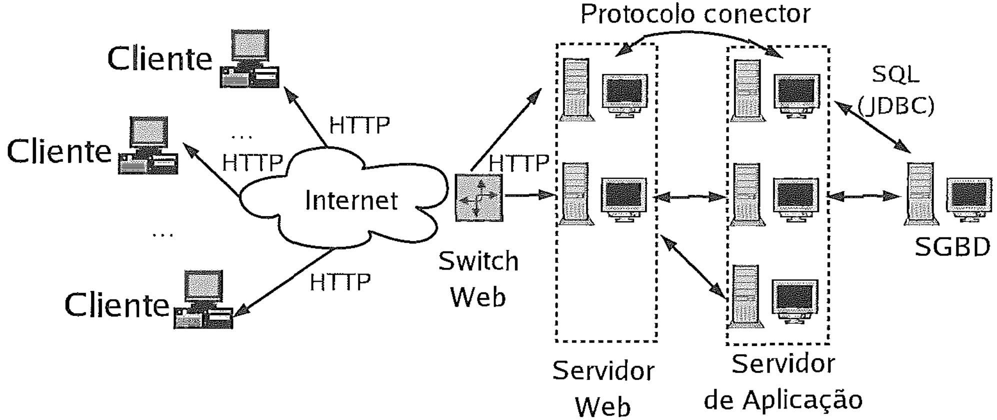

Figura 2.1: Cluster Web

fina sobre os pedidos dos clientes, o comutador Web pode implementar a funcionalidade de um balanceador de carga. Ele pode ser um *switch* em hardware ou uma máquina destinada a esse propósito, e pode também analisar o tráfego Web em múltiplos níveis de rede, incluindo o nível próximo à aplicação (protocolo HTTP), de maneira a decidir para qual nó será encaminhado o próximo pedido.

Servidores de Aplicação

Servidores Web são apropriados para lidar com conteúdo estático, como páginas HTML e imagens residentes no sistema de arquivos desses servidores. As suas implementações são normalmente robustas, eficientes. Por outro lado, o código da aplicação, que é responsável por gerar o conteúdo dinâmico para o cliente, não deve parar um servidor Web, e nem interferir em seu desempenho. Por essa razão, é necessário algum isolamento entre o servidor Web e as aplicações.

Linguagens interpretadas do tipo script, como Perl ou PHP limitam, a possibilidade de um desenvolvedor de aplicação prejudicar o servidor. Entretanto, no caso do servidor Web Apache, o mais popular em uso hoje na Internet [23], os interpretadores Perl e PHP são implementados como módulos embutidos, o que os impede de executar em outra máquina separada do servidor Web e portanto não oferecendo o isolamento completo.

Os servidores de aplicação Java [24, 25, 26, 27] recebem pedidos dos servidores Web, executam o código da aplicação, e devolvem os resultados para os servidores Web. Os servidores Web e de aplicação Java se comunicam por meio de um protocolo conector de rede, o que permite que eles residam em máquinas diferentes, isolando um servidor do outro.

Outra vantagem desses servidores de aplicação Java é que um segundo grau de balanceamento de carga pode ser atingido entre os servidores Web e de aplicação, já que um servidor Web pode encaminhar pedidos a mais de um servidor de aplicação.

Sistemas de gerenciamento de bancos de dados

Finalmente, os dados persistentes são obtidos e armazenados por um sistema de gerenciamento de bancos de dados (SGBD). As consultas e atualizações são realizadas usando-se a linguagem padronizada SQL. No caso de aplicações escritas em Java, a comunicação entre servidores de aplicação e SGBD relacionais é realizada por um *driver* JDBC, como será descrito no capítulo 3.

O acesso ao banco de dados é um importante gargalo no processamento de cargas de trabalho dinâmicas para Web [2, 3]. Entretanto, as operações realizadas por um banco de dados podem se beneficiar do processamento paralelo. SGBD paralelos funcionam como um único SGBD que mantém atomicidade, coerência e isolamento de transações entre várias máquinas garantindo desempenho aceitável para acesso aos dados. O resultado é que SGBDs paralelos são substancialmente mais caros.

2.1.1 Diminuindo o gargalo de acesso ao banco de dados.

Considerando-se as três classes de máquinas paralelas quanto ao compartilhamento de dados - memória compartilhada, disco compartilhado e "nada compartilhado"[1], a arquitetura mais comum para servidores paralelos de bancos de dados é a de "nada compartilhado" [28]. Por essa razão, a vantagem econômica está no uso de clusters que tendam à classe "nada compartilhado", com poucos processadores por nó. Por exemplo, sabe-se que clusters com nós de dois processadores ("2-Way SMP") são mais econômicos que clusters de oito processadores ("8-way SMP"), para uma mesma capacidade agregada de processamento [29].

Entretanto, quanto mais nós há em um cluster, mais mensagens são enviadas entre os processadores, o que significa que uma grande sobrecarga fixa por mensagem é mais significativa. Nesse cenário, é desejável evitar a troca de mensagens quando possível, ou então minimizar a comunicação entre nós, e maximizar a computação dentro dos nós. Bancos de dados paralelos levam isso em consideração, enviando grandes quantidades de dados de uma só vez [28], ou distribuindo com cuidado os dados entre os nós de maneira a minimizar a comunicação.

[1] Na literatura, *shared memory*, *shared disk* e *shared nothing*

Nesse contexto, uma solução para aumentar a vazão de acesso ao banco de dados, e ao mesmo tempo aumentar a disponibilidade, é a replicação de dados entre nós, característica que está disponível em vários SGBDs [30, 31]. Para acessos somente de leitura, a replicação é uma solução trivial, pois qualquer nó pode responder a consultas em qualquer instante, sem comprometer a consistência dos dados. Quando operações de atualização de dados estão envolvidas, é necessário coordenação entre os nós para manter os dados consistentes, o que pode causar queda significativa de desempenho no SGBD. Em vista disso, quanto ao modelo de atualização dos dados [32, 33], a replicação pode ser:

1. Assíncrona - quando as operações de escrita são realizadas em um dos nós e adicionadas a uma fila ou "*log*" para serem propagadas aos outros nós mais tarde (dentro do limite de um intervalo periódico de tempo), depois que uma transação foi consumada[2]. Esse tipo de atualização pode ter um desempenho melhor, mas pode causar inconsistências entre as réplicas, que por sua vez podem originar erros indesejáveis na aplicação. Não é aplicável em todos os casos, portanto.

2. Síncrona - em que os dados são sincronizados com as outras réplicas antes da consumação da transação. Há por isso a garantia de que, ao final de uma transação, os dados estarão consistentes em todas as réplicas. A comunicação é menos compacta, e a desvantagem está no custo, muitas vezes proibitivo, de manter as réplicas sincronizadas. Por isso, o desempenho é em geral pior do que o do modelo assíncrono.

É útil estender essa classificação do modelo de atualização de dados para outras formas de replicação, como por exemplo a replicação de dados entre a cache e o SGBD, ou entre instâncias paralelas da cache. Assim sendo, o projeto do dCache segue o modelo síncrono para a invalidação e atualização da *cache*. O isolamento de transações é respeitado, dentro dos modelos de isolamento "Read Committed" e "Read Uncommited", conforme descrito em 2.3.2.

2.2 Servidores de comércio eletrônico

Um servidor de comércio eletrônico é uma combinação de software e hardware, tipicamente um *cluster Web*, com a função de prover os serviços de uma loja virtual. As interações dos usuários com o servidor de comércio eletrônico podem ser

[2]Depois de ser realizado o *commit*.

representadas por uma seqüência de pedidos que buscam as páginas do servidor. Alguns pedidos possíveis são:

- entrada - pedir a página inicial.
- procurar - procurar produtos por uma palavra-chave.
- navegar - visualizar listas de produtos ofertados.
- selecionar - visualizar um produto em detalhe.
- adicionar às compras - selecionar o produto para ser adquirido
- pagar - identificação do usuário, e eventualmente preenchimento de dados para pagamento e recebimento dos produtos.

A seqüência de pedidos feitos pelo mesmo usuário durante uma mesma visita ao *site* é chamada de "sessão". Esses pedidos resultam no uso dos componentes de hardware e software de um *cluster* para *Web* de maneira não uniforme. Uma visita à página inicial não tem o mesmo peso ao SGBD que, por exemplo, uma procura de produtos por palavra-chave. Por isso, métricas de desempenho como "páginas visitadas por minuto" tendem a não esclarecer os verdadeiros gargalos desse tipo de aplicação.

De forma semelhante, as sessões dos usuários podem ser agrupadas de acordo com diferentes padrões de navegação, que oferecem cargas diferentes sobre o servidor. É possível extrair informações dos *logs* de acesso ao servidor *Web*[3] que revelam, por exemplo, sessões em que os usuários tendem a realizar mais compras e menos pesquisas, ou outras em que os usuários realizam mais pesquisas e menos compras. Separando-se os tipos de sessões, e verificando-se a incidência de cada tipo, pode-se por exemplo analisar e prever, de maneira mais detalhada, o impacto de fatores sazonais no desempenho do sistema, como por exemplo, a proximidade de datas comemorativas em que as vendas superam muito a média de vendas do ano.

Os grafos do modelo de comportamento do cliente (CBMG[4]) para sites de comércio eletrônico, como os da figura 2.2, ajudam a descrever os diferentes grupos de sessões. O grafo A mostra um grupo de sessões em que os usuários tendem a

[3]em [34] e [36] os autores mostram como usar algoritmos de agrupamento (*clustering algorithms*) para obter, a partir dos *logs* do servidor *Web*, grupos de sessões representativos do comportamento dos usuários, com o objetivo de caracterizar a carga em um servidor de comércio eletrônico

[4]*Customer Behavior Model Graph,* como foram propostos em [34]

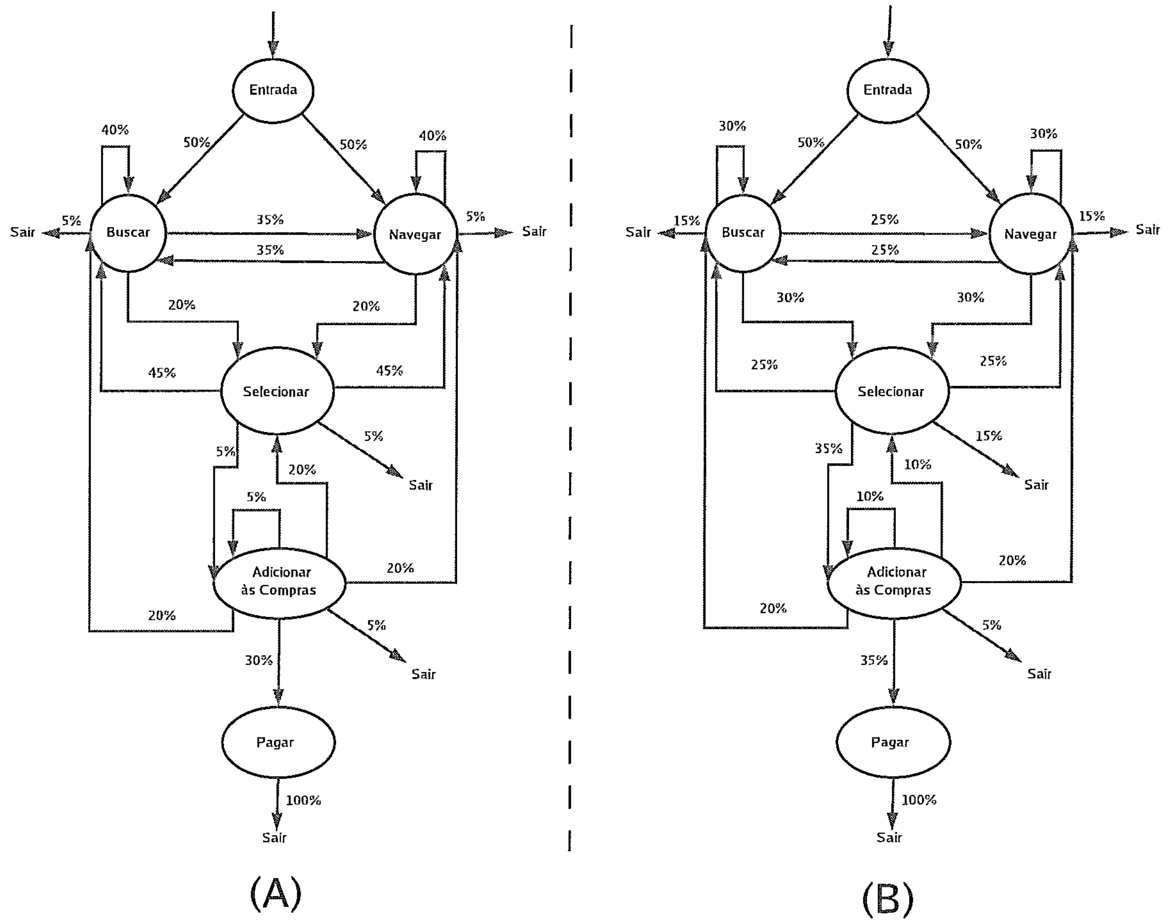

Figura 2.2: Customer Behavior Model Graph

navegar mais e pesquisar mais produtos, mas raramente compram algum produto. O grafo B representa os usuários que têm maior probabilidade de comprar algum produto. Os estados representam as páginas que os usuários pedem, e as arestas direcionadas representam as probabilidades de os usuários passarem de um estado para outro, ou seja, a probabilidade de escolha da próxima página.

Para cada grafo, é possível calcular a quantidade média de visitas a cada estado, por sessão. Numerando-se de 1 a n cada estado, sendo V_1 o número de visitas ao estado de entrada, e V_n ao estado final ("sair"), e considerando-se $p_{k,j}$ a probabilidade de o usuário fazer a transição do estado k para o estado j, tem-se:

$$V_1 = 1$$

$$V_j = \sum_{k=1}^{n-1} V_k \times p_{k,j} \qquad j = 2, \cdots, n-1$$

$$V_n = 1$$

	Entrada	Buscar	Navegar	Selecionar	Adicionar	Pagar	Sair	Total da Sessão
A	1,00	7,83	7,83	3,17	0,17	0,05	1,00	21,05
B	1,00	2,04	2,04	1,27	0,49	0,17	1,00	8,01

Tabela 2.1: Quantidade média de visitas a cada estado para as sessões dos grafos A e B da figura 2.2.

Para cada sessão, a quantidade média de visitas ao estado de entrada e ao estado final é 1. As equações formam um sistema linear que, se resolvido para os valores dos grafos A e B, resultam nos valores da tabela 2.1.

Pode-se verificar a partir desses valores que as sessões do grupo de usuários A - os que mais pesquisam e menos compram, têm um peso maior sobre os estados "Buscar", "Navegar" e "Selecionar" do que os usuários do grupo B. Esses estados implicam operações somente de leitura ao banco de dados, e beneficiam-se mais do uso de *cache*, como é o caso do dCache. Em contrapartida, as sessões do grupo de usuários B têm mais peso do que as do grupo A sobre os estados "Adicionar" e "Pagar", que implicam operações de leitura e escrita ao banco de dados. Tanto em um grupo como no outro as operações de leitura predominam.

Algumas métricas interessantes derivam desse tipo de análise. A soma da quantidade média de visitas a cada página ou estado é a quantidade média de páginas visitadas por sessão, ou tamanho da sessão, representado na tabela 2.1 como "Total da Sessão". A freqüência de visitas ao estado "Pagar" pode ser interpretado como a taxa de sessões que resultam em compras - uma informação útil para propósitos comerciais. A tendência de que quanto mais longa a sessão, menor a probabilidade de que o usuário realize alguma compra, também pode ser verificada pelos dados da tabela.

Os valores das probabilidades de transição escolhidos para os exemplos dos grafos A e B são semelhantes aos apresentados como sendo exemplos típicos por Menascé et al. [34, 36]. Entretanto, esses não são valores reais. Tanto esses autores quanto Wang et al. [37] argumentam sobre a dificuldade ou impossibilidade de se obter dados de sites reais de comércio eletrônico, uma vez que se trata de informação estratégica para os detentores desses dados, que não querem expor o seu negócio.

Existe portanto pouca liberdade para se obter informação real sobre a caracterização de carga em servidores de comércio eletrônico, necessária para a geração de carga para se avaliar os benefícios de um componente novo, hardware ou software, para esse tipo de sistema. Mas essa dificuldade é compensada pela existência de um benchmark específico para se medir a capacidade se um servidor de comércio eletrônico. Esse é o assunto abordado a seguir em 2.2.1.

	Browsing Mix	Shopping Mix	Ordering Mix
Leitura	97,35%	92,36%	63,87%
Leitura/Escrita	1,53%	3,89%	23,02%
Sem uso do B.D.	1,12%	3,75%	13,11%

Tabela 2.2: Proporção das interações com operações de leitura, leitura e escrita, e que não envolvem o uso do banco de dados, conforme a carga.

2.2.1 O *Benchmark* TPC-W

O *benchmark* TPC-W [38], definido pelo Transaction Processing Performance Council[5], simula a atividade de um *site* de comércio eletrônico, especificamente uma loja virtual de livros, em um *cluster* Web. São especificadas 14 interações, realizadas por clientes usando navegadores, sendo que 9 envolvem o banco de dados em operações somente de leitura, 3 envolvem leitura e escrita e 2 não envolvem acesso a banco de dados.

O esquema do banco de dados usado nesse *benchmark* é definido de acordo com o diagrama de entidade e relacionamento da figura 2.3. As tabelas "orders" e "order_line" armazenam dados sobre as compras realizadas, juntamente com "cc_xacts", que representa o uso do cartão de crédito pelos clientes. As tabelas "customer", "address" e "country" armazenam dados sobre os clientes, e as tabelas "item" e "author" têm respectivamente os dados sobre os livros vendidos e os seus autores.

A figura 2.4 mostra as interações do benchmark, e descreve as operações realizadas em cada uma. Conforme a figura, as interações que envolvem apenas consultas, ou leituras, ao banco de dados são o acesso à *home page* (1), a listagem dos 50 livros mais vendidos (11), listagem dos 50 últimos lançamentos (10), o pedido (8) e o resultado (9) da busca de livros por palavra-chave de um determinado autor, título, ou assunto, a exibição para um cliente de seu último pedido (7), a exibição das informações detalhadas de um livro (12), a escolha de um livro para ser atualizado pelo administrador do sistema (13) e finalmente a atualização dos dados do carrinho de compras (2). As interações que envolvem alterações no banco de dados são o pedido de compras pelo cliente (4), a confirmação do pedido de compras (5) e a confirmação da alteração dos dados de um livro pelo administrador (14).

Ainda na mesma figura são mostradas as tabelas usadas em cada operação, e se são usadas para consultas (L) ou para alterações (E). As interações que envolvem transferência segura de dados entre o servidor e o navegador, que devem ser im-

[5]http://www.tpc.org

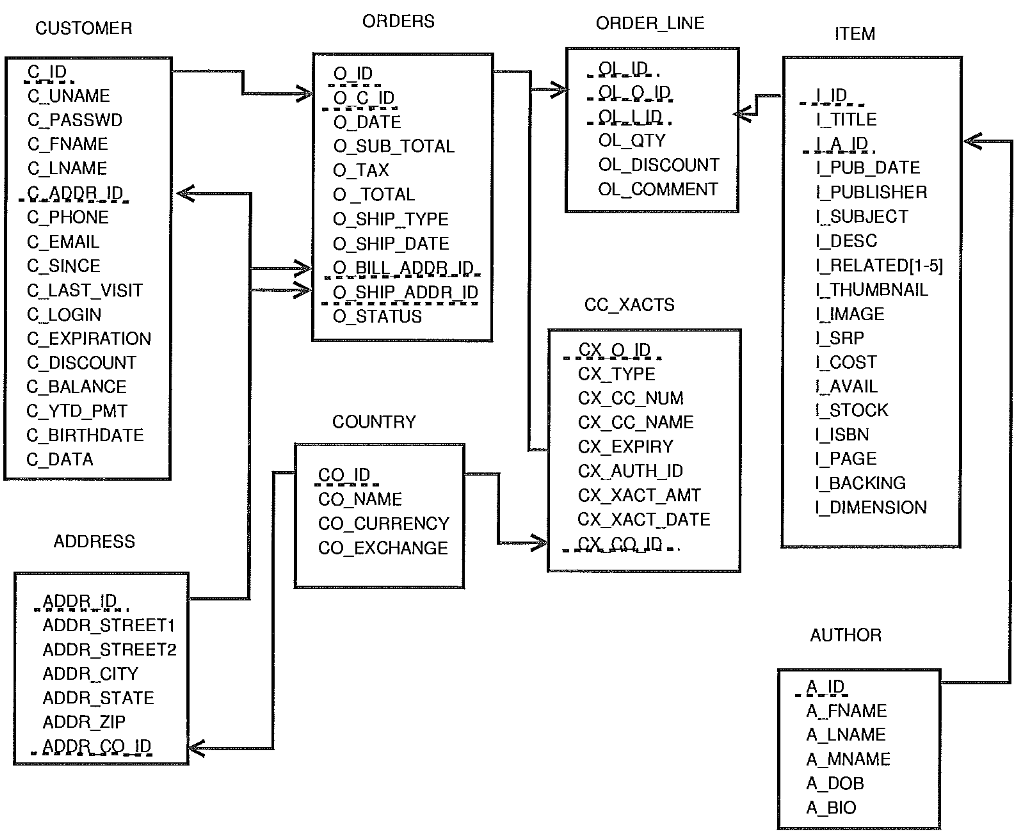

Figura 2.3: Schema do banco de dados do TPC-W

plementadas com os protocolos criptográficos SSL[6] ou TLS[7], são assinaladas com a figura de uma chave. Em uma sessão do usuário, para cada interação existem as outras interações que podem sucedê-la (coluna "próximas interações possíveis") ou precedê-la, (coluna "interações anteriores possíveis"). A partir dessa informação, pode-se construir um diagrama de estados, ou o CBMG do TPC-W. O estado inicial do CBMG é a interação de acesso à *home page* (1).

As arestas com as probabilidades de transição do CBMG dependem do gerador de carga utilizado. A carga é gerada por emulação de navegadores remotos, ou *remote browser emulators* (RBE). Trata-se de um processo em uma máquina separada que utiliza várias linhas de execução para simular a ação de vários usuários acessando o servidor Web via *sockets* TCP/IP. Cada navegador é chamado de *emulated browser* (EB).

A tabela 2.3 mostra as três cargas difententes geradas pelas emulações de navegadores, conforme a especificação do TPC-W. As cargas representam três CBMGs que se distinguem pela proporção entre interações, indicando três grupos de usuários com tendências distintas:

- O Browsing Mix é o grupo de usuários que prefere selecionar os links oferecidos a realizar procuras, e quase não efetua compras.
- O Shopping Mix é o grupo que tende a utilizar as páginas de pesquisa para encontrar os livros que deseja adquirir, e realiza mais compras que o grupo anterior.
- O Ordering Mix, ou OLTP mix, é o grupo de usuários que tende a realizar pesquisas com o propósito específico de encontrar um produto e comprá-lo.

Comparando-se as informações das três cargas da tabela 2.3 com as informações sobre leitura e escrita da figura 2.4, chega-se à proporção de interações de leitura e escrita da tabela 2.2, conforme a carga. Essa informação evidentemente nada diz sobre a extensão de cada operação sobre o banco de dados, e portanto não define a carga real de cada interação sobre o banco, mas aponta a tendência de haver bem mais carga gerada por leituras do que por escritas.

2.2.1.1 Cálculo da vazão no TPC-W

O cálculo da vazão do serviço no TPC-W é denominada WIPS (Web Interactions Per Second), ou número de interações web por segundo. Existe uma denominação

[6] *Secure Sockets Layer*
[7] *Transport Layer Security*

Número da Interação Web	Nome da Interação	Leitura (L) ou Leitura e Escrita (LE)	Atividades	Tabelas envolvidas (L/LE)	Próximas interações possíveis	Interações anteriores possíveis
1	Home	L	Obtém promoções	ITEM (L)	2,6,8,10,11	2,3,4,5,6,7,8,9, 10,11,12,13,14
2	Shopping cart	L	Obtém promoções Atualiza carrinho de compras	ITEM (L) ITEM (L)	1,2,3	1,3,4,8,9,10,11,12
3	Customer registration	-	Usuário inicia autenticação ou começa a registrar-se	-	1,4,8	2
4	🔑 Buy Request	L/E	Obtém dados do cliente Atualiza expiração do login Se o cliente é novo, insere dados do novo cliente Atualiza carrinho de compras	CUSTOMER (L), ADDRESS (L), COUNTRY (L) CUSTOMER (E) CUSTOMER (E), ADDRESS (E), COUNTRY (L) -	1,2,5	3
5	🔑 Buy confirm	L/E	Obtém dados do carrinho de compras Insere novo endereço, se necessário Insere um novo pedido, atualizando o estoque Obtém autorização do PGE Insere dados do cartão de crédito Esvazia carrinho de compras	- ADDRESS (L/E), COUNTRY (L) ORDERS (L/E), ORDER_LINE (E), ITEM (L/E) ADDRESS (L), CC_XACTS (E) -	1,8	4
6	🔑 Order inquiry	-	Usuário identifica-se para solicitar informações sobre seu ultimo pedido	-	1,7,8	1
7	🔑 Order display	L	Obtém senha para autenticar o usuário Obtém último pedido do usuário	CUSTOMER (L) CUSTOMER (L), ORDERS (L)	1,8	6
8	Search request	L	Obtém promoções Usuário escolhe campo e digita chave de busca	ITEM (L) -	1,2,9	1,3,5,6,7,9,10, 11,12,13,14
9	Search result	L	Obtém promoções Conforme o campo de busca escolhido realiza uma das 3: - busca por autor - busca por título - busca por assunto	ITEM (L) ITEM (L), AUTHOR (L)	1,2,8,12	8
10	New products	L	Obtém promoções Obtém livros mais recentes	ITEM (L) ITEM (L), AUTHOR (L)	1,2,8,12	1
11	Best sellers	L	Obtém promoções Obtém livros mais vendidos	ITEM (L) ITEM (L), AUTHOR (L), ORDER_LINE (L)	1,2,8,12	1
12	Product detail	L	Obtém dados detalhados de um livro	ITEM (L), AUTHOR (L)	1,2,8,12,13	9,10,11,12
13	Admin request	L	Obtém dados de um livro para ser atualizado pelo administrador	ITEM (L), AUTHOR (L)	1,8,14	12
14	Admin confirm	L/E	Obtém dados de um livro que foi atualizado pelo administrador Atualiza os dados do livro	ITEM (L), AUTHOR (L) ITEM (E), ORDERS (L), ORDER_LINE (L)	1,8	13

🔑 Transações seguras, usando SSL ou TLS

Figura 2.4: Interações do TPC-W

Web Interaction	Browsing Mix (WIPSb)	Shopping Mix (WIPS)	OLTP Mix (WIPSo)
Browse	95,00%	80,00%	50,00%
Home	29,00%	16,00%	9,12%
New Product	11,00%	5,00%	0,46%
Best Seller	11,00%	5,00%	0,46%
Product Detail	21,00%	17,00%	12,35%
Search Request	12,00%	20,00%	14,53%
Search Results	11,00%	17,00%	13,08%
Order	5,00%	20,00%	50,00%
Shopping Cart	2,00%	11,60%	13,53%
Customer Reg.	0,82%	3,00%	12,86%
Buy Request	0,75%	2,60%	12,73%
Buy Confirm	0,69%	1,20%	10,18%
Order Inquiry	0,30%	0,75%	0,25%
Order Display	0,25%	0,66%	0,22%
Admin Request	0,10%	0,10%	0,12%
Admin Confirm	0,09%	0,09%	0,11%

Tabela 2.3: Proporções das interações do TPC-W conforme os três tipos de carga.

para cada tipo de carga: WIPSb é a vazão para a carga "Browsing Mix", "WIPS" é a vazão para "Shopping Mix", e WIPSo é a vazão para a carga "OLTP Mix".

Existem ainda dois fatores de escala:

- o número de registros da tabela item, que pode variar entre mil, dez mil, cem mil, um milhão e dez milhões de itens, acompanhado pelo número de registros da tabela author, igual a 25% do número de registros da tabela item
- o número de navegadores simultâneos, que define o número de registros da tabela customer em 2880 vezes o número de navegadores, e por conseqüência o número de registros de address em 2 vezes os registros de customer, orders em 90% dos registros de customer, order_line em 3 vezes o número de orders, e cc_xacts com o mesmo número de registros de orders.

Os resultados do benchmark precisam ser relatados em função do fator de escala do número de itens, como por exemplo WIPS@10.000 ou WIPSb@100.000.

O cálculo da vazão é realizado medindo-se intervalos de tempo chamados de WIRT (Web Interaction Response Time), ou tempo de resposta da interação web. Cada WIRT corresponde à diferença de tempo entre o instante em que o navegador emulado envia a primeira requisição HTTP para uma interação web, e o instante em que o navegador recebe a última resposta da última requisição que atende a essa interação. A primeira requisição busca tipicamente o código HTML principal da página, e a última requisição refere-se tipicamente a uma imagem dentro dessa página.

Para o cálculo final são considerados todos os tempos de resposta de todas as interações realizadas pela totalidade dos navegadores emulados dentro de período de medida. Se $WIRTi$ é o tempo de resposta para a interação i, e as interações variam de 1 a N, então,

$$WIPS = \frac{N}{\sum_{i=1}^{N} WIRTi}$$

Entre uma interação e outra, os navegadores simulam um tempo de espera chamado "Think Time", que corresponde ao tempo que um usuário real estaria pensando ou lendo a página, antes de tomar a decisão de escolha da próxima interação. Esse tempo de espera, que é aleatório e segue uma distribuição exponencial negativa, naturalmente não é considerado no cálculo de vazão, e existe somente para tornar a carga mais semelhante a uma carga real.

2.3 *Drivers* JDBC

O JDBC (*Java Database Connectivity*) é o padrão que fornece à linguagem de programação Java a possibilidade de acesso universal a dados. Um fabricante de SGBD em geral fornece um *driver* JDBC para o seu servidor de banco de dados, dessa forma possibilitando que aplicações baseadas em Java acessem e modifiquem dados do servidor, geralmente usando o próprio protocolo de comunicação de rede do SGBD.

Do ponto de vista do desenvolvedor de aplicações, um *driver* JDBC é a implementação da interface de programação de aplicações (API) do JDBC, que permite que se escreva uma aplicação que funcione em qualquer plataforma Java, usando qualquer servidor de bancos de dados relacional que tenha um *driver* JDBC disponível, o que torna a disseminação de informação fácil e econômica entre diferentes plataformas [35]. Em caso de troca do SGBD, não é necessária qualquer modificação ao código da aplicação, desde que apenas comandos dentro do padrão SQL sejam enviados ao servidor de bancos de dados. O dCache, sendo também um *driver* JDBC, não requer modificações ao código fonte da aplicação.

Resumidamente, uma aplicação usa um *driver* JDBC em três passos:

1. Estabelecendo uma conexão com a fonte dos dados[8] (SGBD);
2. Enviando consultas e/ou atualizações à fonte dos dados e
3. Processando os resultados que vêm da fonte de dados.

A figura 2.5 exemplifica esses três passos, do ponto de vista da aplicação, com um fragmento de código, para o caso de uma consulta simples. No primeiro passo, uma string em formato de URL é fornecida para se obter uma conexão com o SGBD, e opcionalmente, são fornecidos dados para a autenticação com o SGBD. A URL informa o nome do *driver* a ser usado, a localização do SGBD na rede, e a base de dados.

No segundo passo, é enviada ao SGBD uma string que corresponde a uma consulta em SQL. No terceiro passo, um cursor itera sobre as tuplas da tabela ou relação resultante da consulta, e são obtidas referências aos componentes de cada tupla, que são por sua vez transferidas para variáveis, obedecendo-se ao tipo de cada atributo, conforme a tabela de exemplo da mesma figura.

[8] O padrão JDBC não restringe a fonte dos dados a um SGBD. Pode-se por exemplo implementar um driver JDBC para acesso a dados em memória RAM ou a arquivos texto, desde que a interface de programação seja respeitada.

```
Connection con = DriverManager.getConnection(
                    "jdbc:someDriver:someDatabase",
                    "user","password");

Statement stmt = con.createStatement();
ResultSet rs = stmt.executeQuery(
                    "SELECT a, b, c from Table1");

while (rs.next()) {
  int x = rs.getInt("a");
  String s = rs.getString("b");
  float f = rs.getFloat("c");
}
```

a	b	c
1	maçã	3,27
2	laranja	1,07
3	melão	2,52
4	banana	2,06

Figura 2.5: Três passos básicos que um desenvolvedor de aplicação realiza para acessar dados por meio de JDBC

2.3.1 Consultas preparadas parametrizadas

Via de regra, as aplicações com interface Web recebem parâmetros de consulta dos usuários e constroem consultas SQL a partir delas. Nesses casos, existe um grupo reduzido de consultas com estrutura semelhante entre si, sendo que entre elas são variados apenas os parâmetros, que são construídos a partir das informações de consulta fornecidas pelos usuários.

Para evitar o custo adicional por parte do SGBD de realizar o *parsing* e de compilar o plano de execução para cada consulta SQL, a API do JDBC, assim como outras API de acesso a bancos de dados, permite que se faça a execução preparada de consultas, em duas etapas. Na primeira etapa, que é feita apenas uma vez, é enviada a estrutura básica da consulta, em que os parâmetros que variam não são informados. O SGDB prepara o plano de execução e retorna um identificador desse plano. Na segunda etapa, são enviados apenas os parâmetros da consulta, juntamente ao identificador de plano de execução fornecido pelo SGBD para aquele tipo de consulta. Como a segunda etapa nas aplicações com interface Web é realizada tipicamente repetidas vezes, economiza-se o tempo de *parsing* e de compilação, assim como também é reduzido o tamanho da mensagem de consulta entre o *driver* e o SGBD, pois a consulta completa em SQL não precisa ser enviada.

```
PreparedStatement statement = con.prepareStatement(
                    "SELECT a,b,c FROM Table1 WHERE b = ?");

// Primeira consulta
statement.setString(1, "laranja");
ResultSet rs = statement.executeQuery();
if (rs.next()) {
  float f=rs.getFloat("c");
}
rs.close();

// Segunda consulta
statement.setString(1, "banana");
rs = statement.executeQuery();
if (rs.next()) {
  float f=rs.getFloat("c");
}
rs.close();

// Terceira consulta
statement.setString(1, "abacate");
rs = statement.executeQuery();
if (rs.next()) {
  float f=rs.getFloat("c");
}
rs.close();
```

Figura 2.6: Consultas preparadas parametrizadas

A figura 2.6 exemplifica, com um trecho de código, como isso é feito do ponto de vista do programador da aplicação. Mesmo que o programador, por engano ou conveniência, tente realizar a primeira etapa mais de uma vez, o *parsing* do SQL já terá sido realizado e o plano de execução já terá sido compilado, e portanto essa tarefa não será refeita e não haverá custo adicional a partir da segunda execução.

Evidentemente é possível com a API do JDBC realizar mudanças no banco de dados, como inserções, atualizações e remoções de dados[9]. Essas operações também podem se valer da execução preparada parametrizada, conforme é mostrado na figura 2.7. Nessa mesma figura também é exemplificado o uso de transações, uma característica de SGBDs que permitem, entre outras características, isolar acessos concorrentes.

Conforme a mesma figura, pode-se informar ao *driver* se cada comando SQL será ou não tratado isoladamente como uma transação (*autocommit*). É possível

[9]Em SQL: INSERT, UPDATE e DELETE

também informar o grau de isolamento das transações, conforme será visto logo adiante em 2.3.2.

As várias operações de escrita e/ou leitura realizadas durante a transação são realizadas em grupo, e de maneira atômica. Ou todas são realizadas ou nenhuma delas é. A transação pode ser consumada (*commit*), e neste caso todos os comandos SQL da transação são aceitos e têm efeito, ou desfeita (r*ollback*), caso em que todos os comandos da transação são rejeitados. Uma transação, com efeito ou não, é delimitada por dois comandos de *commit* ou *rollback*. Logo após o uso desses comandos, os *locks* que protegem os dados contra escrita ou leitura fora de ordem são liberados pelo SGBD.

2.3.2 Isolamento de transações

Os primeiros padrões SQL definiam que o efeito da execução das transações deveria ser como se elas fossem realizadas de forma serial [22]. A API do JDBC permite que se defina o grau de isolamento entre transações descrito pela norma ANSI SQL-92, que é mais flexível, e permite que o programador ajuste o grau de isolamento conforme os requisitos de consistência das transações, alterando potencialmente o grau de concorrência entre elas de forma a aumentar a vazão.

Os graus de isolamento definem o comportamento das consultas, ou leituras, efetuadas dentro das transações. As inserções, atualizações e remoções se comportam sempre da mesma forma. Mas conforme o grau de isolamento, os seguintes fenômenos podem ou não ocorrer com respeito às leituras:

- Leituras sujas[10]: Ocorrem quando em uma transação é lido um atributo com um valor atualizado por uma outra transação em curso, mas essa segunda transação é cancelada (*rollback*), e o valor lido pela primeira passa a não ter mais validade.

- Leituras que não se repetem[11]: Em uma primeira transação é lido o valor de um atributo, e em seguida esse valor é alterado por uma operação de escrita em uma segunda transação. Na primeira transação precisa-se então ler novamente o valor, e quando isso é feito obtém-se um valor diferente do lido anteriormente: o valor alterado pela segunda transação. Duas leituras consecutivas de um mesmo atributo portanto resultam em dois valores diferentes.

[10] Dirty read
[11] Non-repeatable reads

```
con.setAutoCommit(false); // comandos SQL não serão tratados
                  // individualmente como transações

con.setTransactionIsolation(Connection.TRANSACTION_SERIALIZABLE);
                  // grau de isolamento das transações


PreparedStatement statement = connection.prepareStatement(
          "UPDATE Table1 SET c = ? WHERE a = ?");

statement.setFloat(1, 3.59); // atualizar c=3.59
statement.setInt(2, 1);      // p/ a=1
statement.executeUpdate();

statement.setFloat(1, 1.17); // atualizar c=1.17
statement.setInt(2, 2);      // p/ a=2
statement.executeUpdate();

statement.setFloat(1, 2.77); // atualizar c=2.77
statement.setInt(2, 3);      // p/ a=3
statement.executeUpdate();

statement.setFloat(1, 2.26); // atualizar c=2.26
statement.setInt(2, 4);      // p/ a=4
statement.executeUpdate();

erro = validaDados();
if( erro ) {                 // Em caso de erro,
     connection.rollback(); //  desfazer a transação
} else {
     connection.commit();  // caso contrário, confirmar atualizações
}
```

Figura 2.7: Atualizações e transações

- Leituras fantasmas[12]: O fenômeno é semelhante ao de leituras que não se repetem. Em uma primeira transação é lido um atributo, e sem seguida, o registro correspondente ao valor é removido em uma segunda transação. Na primeira transação tenta-se ler o atributo mas o registro não existe mais.

Dados esses fenômenos, os graus de isolamento ANSI previstos no padrão JDBC são:

Read Uncommited

Corresponde ao nível 0 do padrão ANSI. Nesse grau, são permitidas leituras sujas, leituras que não sem repetem e leituras fantasmas.

Read Commited

Nivel 1 do padrão ANSI. Não são permitidas leituras sujas, mas leituras que não se repetem e leituras fantasmas podem ocorrer.

Repeatable Read

Nível 2 do padrão ANSI. Não são permitidas leituras sujas nem leituras que não se repetem, mas leituras fantasmas podem ocorrer.

Serializable

Nível 3 do padrão ANSI. Não podem ocorrer leituras sujas, nem leituras não repetíveis, e nem leituras fantasmas.

Normalmente, um *driver* JDBC limita-se a repassar o grau de isolamento escolhido pelo programador ao SGDB. Como o dCache, entretanto, armazena valores internamente, e tal qual o SGBD precisa decidir o que fazer com eles na presença de transações, o suporte a dois graus de isolamento - "Read Uncomitted" e "Serializable" - foi embutido no *driver*, conforme será visto em 3.

2.3.3 Acesso a metadados

A API do JDBC provê ao programador a possibilidade de acesso aos metadados do banco de dados, como por exemplo a descrição da estrutura das tabelas, chaves primárias e estrangeiras. Também é possível ter acesso a informações específicas sobre as relações resultantes das consultas, como por exemplo o número de colunas

[12]Phantom reads

resultante de uma consulta, ou os rótulos e tipos das colunas. Outras funcionalidades interessantes também são a possibilidade de se obter informações específicas a respeito do SGBD, como por exemplo as palavras-chave suportadas por ele que diferem do padrão SQL, tipos nativos que ele adota, a relação das funções que são aceitas pelo SGBD (como as matemáticas, de manipulação de strings etc) [35].

O provimento de metadados é de responsabilidade do implementador do *driver* de cada SGBD. Assim, há garantia de que os metadados estarão disponíveis de forma padronizada para o programador de aplicação que utiliza a API JDBC, independentemente da forma de obtenção desses metadados para cada SGBD em particular.

O acesso a metadados é algo pouco utilizado pelos programadores de aplicações, mas é útil para quem implementa utilitários de gerência de bancos de dados e implementadores de *drivers*, principalmente os *drivers* implementados como ponte sobre outros *drivers*, como é o caso do dCache. Essa característica foi utilizada da implementação do dCache principalmente para obtenção do esquema (ou *schema)* do banco de dados usado pela aplicação, e para identificação das chaves primárias, usadas para localizar os dados na cache, e para atualizar esses dados no caso de operações de escrita.

2.3.4 Tipos de *drivers* JDBC

Os *drivers* JDBC são tradicionalmente divididos em 4 tipos [35, 39]:

1. Tipo "ponte JDBC-ODBC": Esse tipo de *driver* traduz as chamadas à API JDBC para a API ODBC[13]. É usado quando não existe um *driver* JDBC específico para determinado SGBD, mas há disponível o *driver* ODBC. A desvantagem é o fato de que se trata de uma indireção dupla, pois é obrigatório que sejam usados os dois *drivers*, além da biblioteca binária de acesso ao SGBD, fornecida pelo fabricante, que também precisa estar instalada na máquina.

2. Tipo "Parte Java, parte API nativa": É escrito de forma a fazer uso direto da biblioteca binária de acesso ao SGBD fornecida pelo fabricante (que normalmente tem uma interface para a linguagem C) instalada na máquina cliente, que faz o acesso ao SGBD via o protocolo de rede próprio deste último. Apesar de oferecer um desempenho melhor do que os *drivers* de tipo 1, ainda apresenta a inconveniência de não ser totalmente escrito em Java, o

[13] Open DataBase Connectivity, da Microsoft

que dificulta o seu uso em aplicações que têm "código móvel" (instalados sob demanda), como é o caso das "applets" do Java na Internet.

3. Tipo "Escrito totalmente em Java, com protocolo de rede de *middleware*": Para eliminar a necessidade de se ter instalada a biblioteca binária de acesso ao SGBD fornecida pelo fabricante na máquina do cliente, o modelo de três camadas[14] é empregado nesse tipo de implementação de *driver* JDBC. A biblioteca binária (ou também possivelmente o *driver* ODBC) é instalada apenas na camada do meio, ou *middleware*. Os clientes usam um *driver* JDBC que faz acesso via rede à camada do meio, onde existe um *driver* do tipo 1 ou 2, que por sua vez faz o acesso ao SGBD. Esse modelo permite que se empreguem as técnicas de balanceamento de carga e de *cache* entre os vários clientes, pois os acessos são centralizados no *middleware*.

4. Tipo "Escrito totalmente em Java, com protocolo de rede nativo do SGBD". *Drivers* desse tipo convertem as chamadas à API JDBC diretamente no protocolo de rede nativo do SGBD. Por serem totalmente implementados em Java, possibilitam maior flexibilidade e independência de plataforma, podendo ser usados tanto com código móvel (*applets*) como em servidores de aplicação ou arquiteturas cliente-servidor.

A figura 2.8 ilustra essa classificação. Existem ainda *drivers* que não se encaixam perfeitamente em nenhum dos 4 tipos tradicionais dessa classificação. É o caso do middleware C-JDBC [13], destinado a prover a paralelização transparente de SGBD, até mesmo de fabricantes diferentes, que foi apelidada jocosamente de *RAIDb* [15]. O C-JDBC poderia ser classificado como tipo 3 por ser um *middleware*, mas implementa acesso paralelo a diversos bancos por meio de outros *drivers*, inclusive do tipo 4, e a outras instâncias de *drivers* de seu próprio tipo, de maneira hierárquica. Uma descrição desse *driver* é feita juntamente a outros trabalhos relacionados, no capítulo 5.

O dCache também não encontra classificação direta nessas categorias, mas pederia ser classificado como "ponte JDBC-JDBC", pois é um *driver* JDBC que adiciona funcionalidades a outro *driver* JDBC. A aplicação utiliza diretamente o *driver* dCache, ou *driver* "mestre", e quando as requisições não podem ser atendidas pela cache, o dCache utiliza o *driver* específico do SGBD, ou *driver* "escravo", para obter os dados da banco. Os detalhes da arquitetura do dCache são vistos em 3.

[14] Arquitetura "*three-tier*"

[15] "*Redundant Array of Inexpensive Databases*"

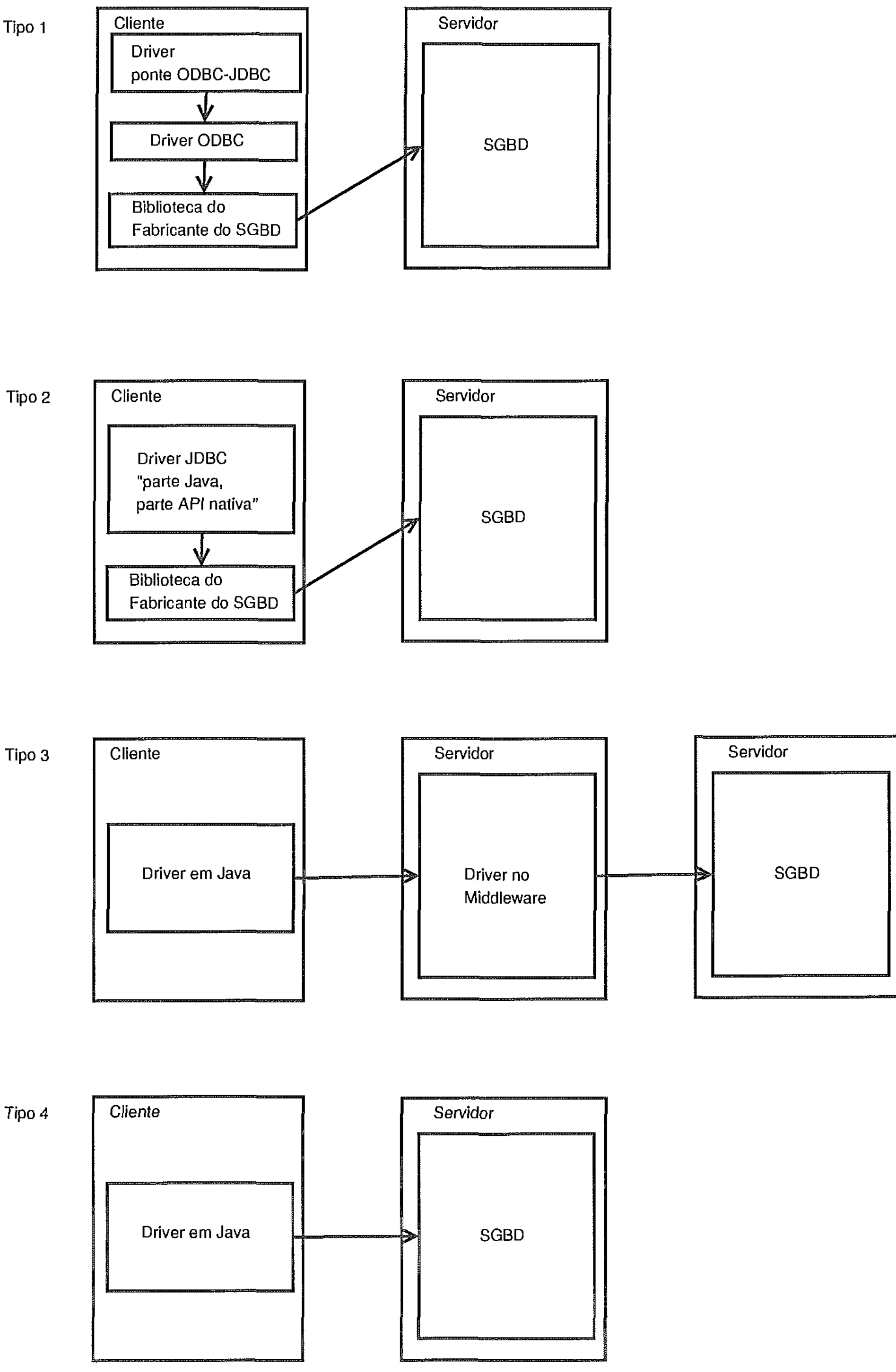

Figura 2.8: Tipos de *driver* JDBC

2.3.5 Gerente de *Drivers*

A API do JDBC prevê o uso de mais de um *driver* simultaneamente. Para isso, fornece uma classe chamada *DriverManager*, que é um gerente dos *drivers*. Essa é uma característica interessante para implementações como o C-JDBC ou o dCache, porque permite que um *driver* utilize o outro de maneira organizada, usando a própria API.

Para que um *driver* seja usado, deve primeiro ser registrado junto ao gerente. Isso é feito no próprio código do *driver*. Quando o programador da aplicação solicita uma conexão a um banco de dados, ele primeiro fornece uma URL que descreve o protocolo a ser usado, contendo a identificação do *driver*, a localização do SGBD na rede, o nome do banco de dados e opcionalmente outras informações, como nome de usuário e senha para autenticação, como no exemplo abaixo:

jdbc:mysql://localhost/tpcw?user=benchmark&password=e2r7529

Essa URL informa ao gerente que o protocolo "jdbc" deve ser usado para conectar-se a um SGBD "mysql" na máquina "localhost", para acessar o banco de dados "tpcw", utilizando como informações de autenticação o nome de usuário "benchmark" e a senha "e2r7529".

O gerente de *drivers* percorre então uma lista em memória de *drivers* disponíveis registrados, entregando a URL a cada um deles, em seqüência. O primeiro *driver* que seja capaz de estabelecer a conexão o faz e a devolve para o gerente, que a repassa para a aplicação.

Esse padrão de programação permite que um SGBD seja trocado por outro trocando-se apenas a URL, e portanto sem que tenha que haver maiores modificações no código da aplicação. A informação a respeito de qual *driver* escravo será utilizado juntamente ao dCache é colocada nessa URL, como será visto a seguir no capítulo 3.

Capítulo 3

A arquitetura do dCache

O *driver* dCache foi desenvolvido com o objetivo de obter ganhos de desempenho em aplicações de comércio eletrônico para a Web, sem que seja necessária a alteração dessas aplicações. São assumidas as seguintes características para essas aplicações:

1. Existe um padrão de acesso que envolve uma taxa maior de dados recebidos por parte dos usuários do que dados enviados, como ocorre normalmente em sistemas para a Web. Isso se reflete em uma quantidade maior de leituras do que escritas no banco de dados.

2. Aplicações de comércio eletrônico envolvem a pesquisa de itens de algum catálogo. Alterações de informações desse catálogo são feitas com freqüência bastante menor do que a freqüência dos acessos feitos a essas informações pelos usuários, e quando seguem o modelo do benchmark TPC-W, são feitas pelos administradores no sistema utilizando uma interação via Web - tipicamente uma interação para cada item alterado.

3. Os usuários dessas aplicações fazem buscas por palavras-chave para encontrar itens que lhes interessam. Essas buscas seguem uma distribuição estatística típica daquelas que regem o comportamento humano em grandes populações, como é o caso da distribuição de Zipf e demais distribuições ditas de cauda pesada, como a distribuição de Pareto. Com isso, mesmo uma cache pequena pode melhorar significativamente o desempenho de um sistema [40].

4. A maior parte das escritas ao banco de dados devidas às interações dos usuários diz respeito a dados que têm pouco impacto global no sistema, e mais impacto nas leituras no banco de dados de interesse do próprio usuário.

Esse cenário definiu o projeto da arquitetura interna do dCache. Além disso, há a premissa importante de que a aplicação não tenha que ser alterada para

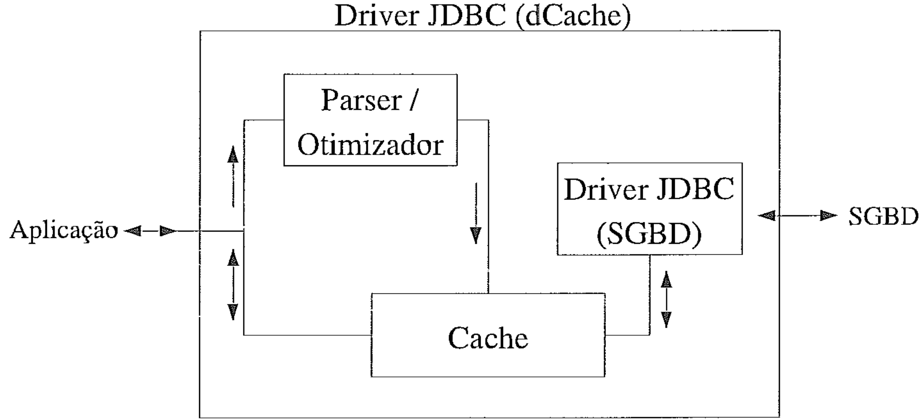

Figura 3.1: Arquitetura do dCache

se beneficiar da cache. Mas para que isso tenha efeito, é necessário que o *driver* tenha a capacidade não só de armazenar os valores na cache de maneira genérica, como também invalidar ou atualizar valores em cache em função das operações de escrita solicitadas ao próprio *driver* pela aplicação, também de maneira genérica.

3.1 Arquitetura do dCache

A figura 3.1 mostra a arquitetura interna do dCache. A aplicação faz uso do dCache como um *driver* JDBC usual, que provê a API padronizada de acesso a banco de dados. O dCache funciona associado ao *driver* JDBC original fornecido pelo fabricante do SGBD, e é este *driver* que realiza toda a comunicação e a obtenção de dados do SGBD.

Os dados obtidos a partir de consultas ficam armazenados em *cache*. Todas as consultas e operações de escrita no banco de dados são solicitadas ao *driver* dCache, de forma transparente para a aplicação, assim como todos os dados obtidos pela aplicação são providos por esse *driver*, que provê esses dados fazendo acesso à *cache* ou aos dados obtidos pelo *driver* do fabricante do SGBD.

Os comandos SQL, tanto de leitura quanto de escrita, quando utilizados pela primeira vez, são analisados por um parser, que organiza esses comandos em estruturas de árvore sintática, que são armazenadas em uma *cache* de comandos, evitando-se assim o custo de *parsing* para comandos anteriormente executados. Além das estruturas de árvore sintática, na *cache* são armazenados dados que permitem o acesso otimizado a informações da consulta utilizadas para a realização

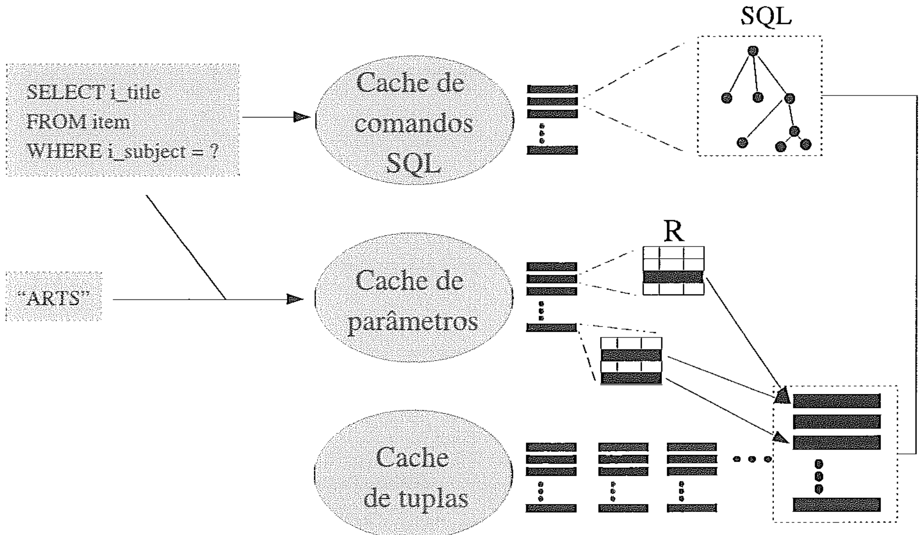

Figura 3.2: Estruturas internas de cache do *driver* dCache

das atualizações e invalidações em cache.

3.1.1 Estruturas de cache

Existem três estruturas de cache distintas no dCache, na forma de tabelas *hash*, conforme ilustrado na figura 3.2. Todas essas tabelas *hash* têm o comportamento de uma cache com política de substituição LRU[1]. As três estruturas são:

1. Cache de comandos SQL (ou *Statement Cache*) - tem como chave de indexação da tabela *hash* um comando SQL (na forma de *string*), e armazena as estruturas de árvore sintática geradas durante o *parsing* do comando SQL.

2. Cache de parâmetros (ou *Param Cache*) - tem como chave de indexação a combinação de um comando SQL e dos parâmetros usados na consulta, e armazena as relações resultantes das consultas.

3. Cache de tuplas (ou *Row Hash*) - tem como chave de indexação as chaves primárias que ocorrem nas tuplas das relações, e armazena as tuplas propriamente ditas, ou seja, os conjuntos ordenados dos valores dos atributos para a tupla. Duas ou mais relações podem ter referências para uma mesma tupla na cache de tupla, evitando-se assim duplicação de dados em *cache*.

[1]Least-Recently-Used: em caso de falta de espaço, é descartado o dado contido na *cache* que não é utilizado há mais tempo para que uma novo tome seu lugar.

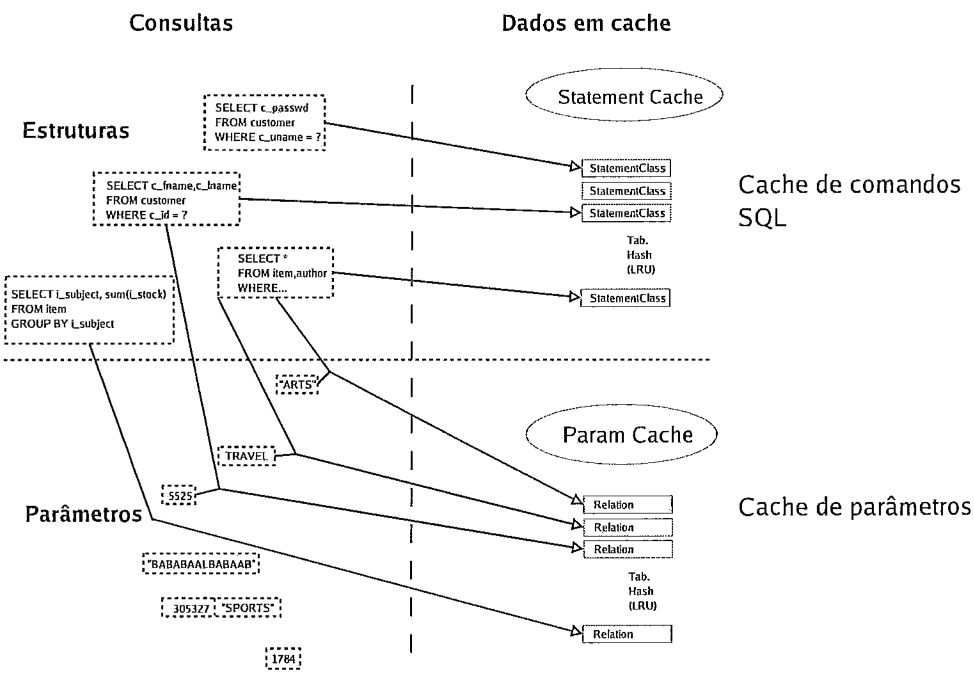

Figura 3.3: Funcionamento da cache de comandos SQL e da cache de parâmetros para consultas.

3.2 Funcionamento básico das consultas

A primeira escolha de projeto foi a de implementar a cache como uma cache de consultas, conforme a definição de modelos de armazenamento do capítulo 1. Pode-se pensar nesse modelo como uma generalização do e-Cache [9], de forma que a consulta SQL do dCache é análoga aos termos de pesquisa no e-Cache.

A figura 3.3 mostra o funcionamento interno do dCache, com relação às estruturas de dados de cache de comandos e cache de parâmetros. A consulta SQL é informada ao *driver* pelo programador da aplicação usando-se a API do JDBC. Se a consulta for parametrizada, conforme 2.3.1, a estrutura da consulta é informada, assim como os parâmetros, quando os houver.

A seguir, a aplicação requisita ao SGBD a execução da consulta. Nesse momento, em vez de enviar a requisição ao SGBD, o *driver* dCache verifica na tabela *hash* da "Statement Cache", ou cache de comandos SQL, se lá já existe uma versão pré-compilada da consulta, versão essa chamada de "StatementClass". Caso não haja, como no caso em que uma estrutura de consulta parametrizada é utilizada pela primeira vez, a compilação é efetuada, pelos motivos descritos mais adiante, e conforme a explicação em 3.3, e a versão compilada é armazenada na "Statement Cache" .

Usando-se a combinação da string da consulta SQL com os parâmetros de con-

Relações

Relation
índice1, índice2...
Vetor Dinâmico
Param Cache
Cache de parâmetros
Relation
Tab. Hash (LRU)
Row Hash
Tupla
Tab. Hash (LRU)
Cache de tuplas
atributo1 atributo2 ... atributo n

Figura 3.4: Relação entre cache de parâmetros e cache de tuplas

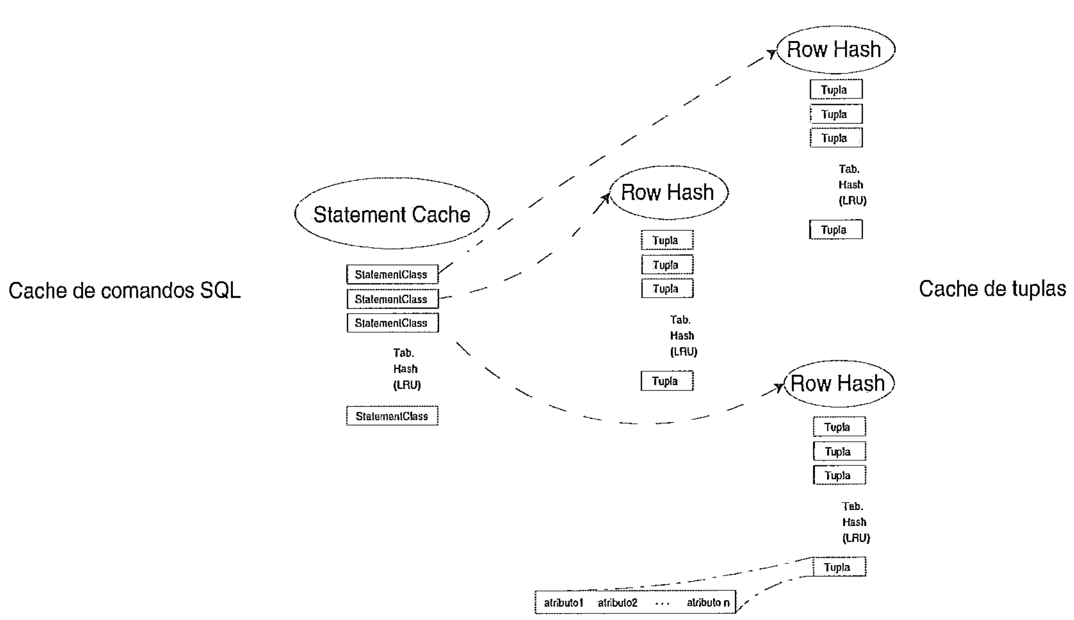

Figura 3.5: Relação entre cache de comandos SQL e cache de tuplas

sulta como chave, é buscada a relação resultante da execução da consulta em uma segunda tabela *hash*, chamada "Param Cache", ou cache de parâmetros, que também tem um comportamento de *cache* LRU. Caso uma relação não seja encontrada nessa busca, uma nova é criada e indexada nessa *cache* pela chave de busca.

A relação é o conjunto de tuplas que representa o resultado da consulta[2]. Na figura 3.4, ela é chamada de "Relation"[3]. Em alguns casos envolvendo consultas com agregações, cada tupla nessa etapa já representa diretamente um vetor com os valores dos atributos para a tupla. No caso mais genérico, a tupla será encontrada em uma última tabela *hash* com comportamento de cache LRU, chamada "Row Hash", ou *cache* de tuplas, usando-se para isso uma combinação das chaves primárias das tabelas envolvidas na consulta.

Existe uma instância da *cache* de tuplas para cada consulta compilada, conforme ilustra a figura 3.5, e o propósito dessa *cache* de tuplas é permitir que mais de uma relação tenha uma referência a uma mesma tupla em *cache* sem que haja replicação de tuplas, conforme ilustrado na figura 3.4.

3.3 Atualizações e invalidações

O *driver* dCache considera a possibilidade de operações de escrita (modificações) ao banco de dados, tomando para isso, durante as consultas, imediatamente antes de acessar o SGBD, providências que possibilitem a invalidação ou atualização dos dados de sua cache. A questão da granularidade da invalidação ou atualização de valores de uma cache que usa o modelo de cache de consultas tem impacto direto no desempenho da cache. Ainda que seja favorável a premissa de que a quantidade de escritas é menor que a de leituras, a opção de invalidar todos os dados em cache a cada operação de escrita é praticamente inviável, pois acarreta uma baixa taxa de acertos *(hits)* em *cache*, e tanto mais baixa é essa taxa quanto maior a taxa de escritas, conforme experimentos de Degenaro et al. [21]. É necessário que haja algum nível de granularidade para atualizações e invalidações na cache.

A opção de granularidade na implementação da dCache foi a de manter uma tupla de uma relação como a unidade mínima na cache, passível de atualização individual em cache pelos comando UPDATE da linguagem SQL, e de invalidar grupos de tuplas afetados pelos comandos INSERT e DELETE para cada tipo de consulta. Sendo assim, é necessário:

[2]Conforme a nomenclatura usual em bancos de dados relacionais [22].

[3]Na figura, os nomes das estruturas, em inglês, são os mesmos adotados nas etapas de projeto e codificação do *driver*.

- Identificar cada tupla em cache afetada por uma operação de escrita UPDATE.
- Identificar o conjunto de tuplas afetadas por uma operação de escrita INSERT ou DELETE.

A identificação do conjunto de tuplas afetadas por INSERT ou DELETE é feita pela identificação das tabelas envolvidas nesses comandos e nas consultas. A identificação individual de cada tupla afetada por UPDATE é feita por meio da chave primária de cada tabela envolvida em uma consulta, chave essa que devido às providências tomadas pelo *driver* vistas a seguir, asseguradamente faz parte da tupla.

3.3.1 Atualizações

A atualização dos valores em cache visa a evitar o custo das invalidações, que significam a ocorrência de falhas em cache e conseqüentes buscas ao banco de dados. As atualizações são feitas em função do uso de comandos UPDATE.

Reescrita das consultas

Nem todas as consultas em SQL resultam em respostas que incluem as chaves primárias das tabelas envolvidas. Por isso, o dCache reescreve as consultas em tempo de execução para incluir todos os índices necessários. Essa reescrita é feita alterando-se o operador de projeção[4] do comando SELECT da seguinte forma. Dada uma relação R, resultante da operação de seleção do predicado de um comando SELECT, e a relação P resultante da projeção sobre essa mesma relação:

$$P = \pi_{A_1,A_2,\ldots,A_n}(R) \ ,$$

e dadas $K_1, K_2, ..., K_m$, as chaves primárias das m tabelas[5] envolvidas na projeção (que são as tabelas a que pertencem os atributos $A_1, A_2, .., A_n$), a alteração consiste em substituir a relação P resultante da primeira projeção por outra relação P' resultante de uma nova projeção:

$$P' = \pi_{A_1,A_2,\ldots,A_n,K_1,K_2,..,K_m}(R) \ ,$$

[4]De acordo com a álgebra relacional, a operação de projeção (π) é usada para produzir a partir de uma relação R uma nova relação que tenha apenas algumas das colunas de R [22]. Assim, a expressão $\pi_{A_1,A_2,\ldots,A_n}(R)$ resulta na relação que tem apenas as colunas correspondentes aos atributos $A_1, A_2, ..., A_n$de R.

[5]Essas tabelas aqui são aquelas que fazem parte do esquema do banco de dados.

sendo $K_i = I_{i,1}, I_{i,2}, ..., I_{i,j}$ os j atributos que integram a chave K_i.

As m tabelas envolvidas nessa operação incluem as tabelas renomeadas (as que têm um *alias*), que são normalmente empregadas em operações de junção (*join*). Isso significa que, entre as chaves $K_1, K_2, ..., K_m$, podem ocorrer chaves que representam fisicamente a mesma tabela no banco de dados, mas que, devido ao *alias*, passam a ser consideradas como distintas, como acontece em operações de *self-join*[6].

Levando-se em conta o esquema do banco de dados representado na figura 2.3, tome-se como exemplo a consulta:

> SELECT i_title, a_lname, i_cost FROM item, author WHERE item.i_a_id = author.a_id

Essa consulta retorna a projeção dos atributos "i_title" e "i_cost" das linhas selecionadas da tabela "item", e do atributo "a_lname" da tabela "author", sobre a relação correspondente à junção (*join*) da tabela "author" com a tabela "item" pela chave estrangeira "item.i_a_id" desta e pela chave primária "author.a_id" daquela. Sem a operação de reescrita da consulta, caso ocorresse uma atualização no banco de dados, como por exemplo,

> UPDATE item SET i_cost=30 WHERE i_id=5342

não haveria como atualizar o atributo "i_cost", pertencente à tupla que reside em cache como conseqüência da consulta anterior, porque não existiria relacionamento, na cache, entre o registro afetado pela atualização, identificado pela chave primária i_id=5342, e a tupla que contém os atributos desse registro. Lembrando que se trata de uma cache de consultas, não restaria alternativa senão invalidar todas as entradas em cache correspondentes à consulta.

Por isso, o dCache intercepta as consultas SQL direcionadas ao SGBD e as modifica para conterem os índices, ou chaves, necessárias, fazendo a substituição de P por P' mostrada anteriormente. Como resultado dessa operação, a consulta enviada ao SGBD pelo dCache seria:

> SELECT i_title, a_lname, i_cost, item.i_id, author.a_id FROM item, author WHERE item.i_a_id = author.a_id

Adicionando-se portanto "item.i_id" e "author.a_id" à operação de projeção da consulta de maneira a produzir uma nova relação que inclua essas duas colunas.

---

[6] O *join* de uma tabela com ela mesma, usada quando se quer conectar atributos que estão em uma mesma tabela

Essas colunas são invisíveis para a aplicação, e são usadas apenas internamente pelo *driver* para os propósitos de indexação das tuplas.

Para que essa modificação à consulta seja feita de forma transparente e sem necessidade de intervenção humana, o *driver* utiliza o acesso a metadados visto em 2.3.3 para conhecer as chaves primárias de cada tabela, chaves essas que obviamente podem ser representadas pela combinação de mais de um atributo da tabela ($K_i = I_{i,1}, I_{i,2}, ..., I_{i,j}$, como visto anteriormente).

Para determinar quais tabelas estão envolvidas na consulta, assim como para reescrever a nova projeção incluindo as chaves de maneira automática, o *driver* faz o *parsing* da consulta em SQL, e assim obtém uma série de objetos representando os elementos sintáticos da consulta de maneira encadeada, em uma árvore sintática. Essa árvore é então armazenada como parte da consulta pré-compilada, ou StatementClass, e esta é por sua vez em seguida posta na "Statement Cache". Com isso, novos acessos que utilizem uma consulta SQL que tenha a mesma estrutura já usada anteriormente podem se valer da consulta pré-compilada sem que haja necessidade de fazer um novo *parsing*.

No apêndice B é mostrada uma interface gráfica que apresenta a possibilidade de se examinar a árvore sintática de uma consulta, e no apêndice A há detalhes sobre a implementação do parser, que foi realizada com o auxílio do Javacc [41], uma ferramenta geradora de parsers.

A reescrita da consulta, assim como o *parsing* dela, faz parte da etapa de compilação da consulta, e é feita apenas uma vez para cada consulta armazenada na "Statement Cache".

Vetor de dependência da projeção.

Durante a compilação de uma consulta, é construído um vetor de dependência entre os atributos usados na operação de projeção dessa consulta. Dado o vetor V_C das m chaves primárias envolvidas na projeção que resulta na relação $P = \pi_{A_1,A_2,...,A_n}(R)$,

$$V_C = < K_1, K_2, ..., K_m >,$$

o vetor de dependência V_{dp} é representado por

$$V_{dp} = < dp_1, dp_2, ..., dp_n >$$

sendo dp_i um subconjunto do conjunto $\{1, 2, ..., m\}$ dos índices das chaves do vetor V_C. O conjunto dp_i representa os índices das chaves em V_C das quais o atributo A_i depende.

Considerando ainda o esquema da figura 2.3, a figura 3.6 mostra um exemplo da criação do vetor de dependência para uma consulta. A consulta é informada ao *driver* (a), e então reescrita de modo a incluir as chaves K_1 e K_2 (c). Após a execução da consulta, o SGBD envia como resposta a relação P', que inclui as chaves para as tabelas envolvidas na consulta (d). O vetor de dependência da projeção V_{dp} é computado (e), indicando que o atributo A_1 depende da chaves em V_C cujos índices fazem parte do conjunto $\{2\}$ (no caso, somente K_2), e que A_2, da mesma forma, também depende de K_2.

Tabelas de dependência global

Logo que o *driver* acessa os metadados para buscar o schema do banco de dados, são criadas tabelas de dependência global para os atributos das tabelas. É criada uma tabela Hash de dependência global para cada atributo de cada tabela do schema do banco de dados. As tabelas de dependência informam o valor mais atualizado de cada atributo, indexado pela chave primária de cada tabela do schema. E elas são chamadas globais porque seu conteúdo pode ser de interesse de qualquer consulta compilada.

Conforme o exemplo visto na figura 3.6, a relação P é a relação percebida pela aplicação (b). Sempre que a aplicação solicitar uma referência aos atributos A_1 ou A_2 de P, o *driver* dCache verifica antes se houve alguma modificação de valor para o atributo em questão nas tabelas de dependência global. Seguindo o exemplo da mesma figura, caso a aplicação solicite o valor do atributo A_2 para a primeira tupla de P, o *driver* verifica primeiro em P' o valor correspondente de K_2 para a tupla - no caso, "907" - e usa esse valor como entrada para buscar um novo valor na tabela de dependência global para o atributo A_2. Caso exista esse valor, ele é retornado pelo *driver* para a aplicação como a referência mais atualizada do atributo A_2. Caso não exista o valor, é retornado o correspondente na tupla armazenada em cache.

As tabelas de dependência global são alteradas quando há alteração de atributos de algum registro no banco de dados devido ao uso do comando UPDATE, e quando essa alteração é determinada pelas chaves primárias, conforme o exemplo abaixo:

UPDATE item SET i_thumbnail = 'img7/thumb_907b.gif' WHERE i_id = 907

Nesse exemplo, a tabela de dependência global do atributo "item.i_thumbnail" mapearia a chave {907} no valor "img7/thumb_907b.gif", que é o valor mais atualizado para esse atributo e essa chave correspondente.

Essa forma de organização dos dados em cache - vetor de dependência e tabelas de dependência global - tem os seguintes objetivos:

- Permitir que a alteração de atributos seja rápida e centralizada. Em vez de se buscar as tuplas em cache para alterá-las, sendo que não se sabe de antemão quais tuplas serão futuramente usadas e se beneficiarão da atualização, a alteração é registrada na tabela de dependência global, e adiada até o último momento - aquele em que a aplicação finalmente solicita o atributo modificado.
- Impedir que a invalidação de um ou mais atributos em uma tupla de uma relação obrigue o *driver* a uma busca ao SGBD, desprezando o benefício da existência de outras tuplas em cache para a mesma relação. A penalidade para se buscar uma tupla invalidada em uma relação é muito próxima da penalidade de se buscar a relação inteira.

Lote de operações de commit

A cada conexão ao banco de dados está associada uma lista chamada "lote de operações". O comando UPDATE não causa a alteração na tabela de dependência global até que a alteração seja confirmada pelo banco de dados após o comando "commit". Só então são realizadas, para uma dada transação, as alterações dos valores em cache, ou seja, nas tabelas de dependência.

As alterações que devem ser realizadas em cache são acumuladas durante uma transação, seqüencialmente em uma lista ou lote de operações. O lote de operações é executado de uma só vez durante o commit, e pode incluir também, além das operações de atualização, operações de invalidação e operações de *lock* e *unlock* das tabelas de dependência global.

Ao final de uma operação de *commit* ou *rollback* de uma conexão, o lote de operações é removido para a conexão em questão. No caso de *commit*, as operações são realizadas, e no caso de *rollback*, apenas descartadas.

3.3.2 Invalidações

As invalidações, que são de implementação bem mais simples que as atualizações, consistem na eliminação de um conjunto de tuplas da cache e causam falhas em cache durante as consultas. São realizadas em função dos comandos INSERT e DELETE.

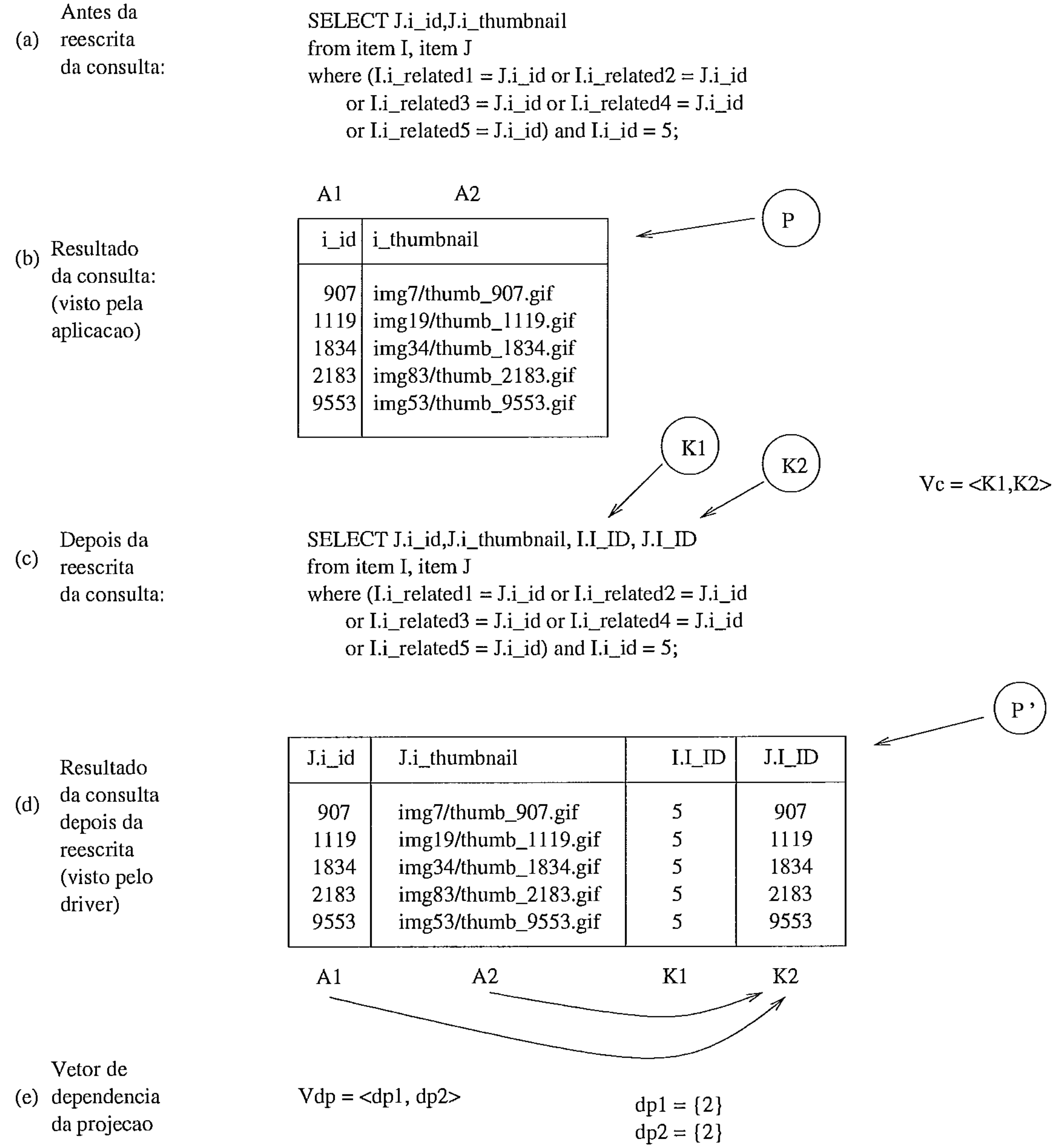

Figura 3.6: Exemplo de construção do vetor de dependência da projeção.

A cada comando INSERT ou DELETE, são acumuladas, no lote de operações, as operações de invalidação referentes às tabelas envolvidas no comando. Para cada tabela do banco de dados, existe um rótulo de tempo (*timestamp*) informando o momento da última operação de INSERT ou DELETE sobre tal tabela. A operação de invalidação consiste em atualizar esse rótulo de tempo para a tabela envolvida.

A invalidação propriamente dita ocorre durante uma consulta. Nesse momento, o rótulo da última invalidação ocorrida para aquele tipo de consulta, que se encontra armazenado na estrutura compilada, é comparado com os rótulos de tempo das tabelas que participam da consulta. Se algum dos rótulos de tempo das tabelas for de valor maior que o rótulo da última invalidação para a estrutura ou tipo de consulta (StatementClass) em questão, então todas as tuplas associadas a esse tipo de consulta são eliminadas da cache de tuplas (RowHash), e o rótulo de tempo para o tipo de consulta é atualizado. O efeito imediato dessa eliminação das tuplas é a ocorrência de falha em cache quando a aplicação solicita uma referência a um valor de atributo de alguma dessas tuplas.

A figura 3.7 ilustra e exemplifica a ocorrência de falhas por invalidação. As falhas por invalidação são apenas um dos tipos de falha possíveis. As falhas também podem ocorrer simplesmente porque a relação ou alguma de suas tuplas nunca pertenceu à cache, ou porque alguma delas foi descartada pela política de substituição da cache, ou ainda porque houve suspensão temporária do uso da cache, como será visto em 3.3.3.

A invalidação tem o efeito indesejável de eliminar as tuplas que foram colecionadas para o tipo de consulta afetado. Entretanto consultas de outros tipos, que não envolvam as tabelas que foram alteradas, são preservadas.

3.3.3 Suspensão temporária do uso da cache

Para garantir que haja um grau adequado de isolamento entre as transações, pode haver a suspensão temporária do uso da cache em determinados momentos. O compromisso do dCache é o de manter o grau de isolamento *read commited* (nível 1 do padrão ANSI SQL-92), impossibilitando leituras sujas, ou seja, de permitir que a cache forneça apenas dados cuja atualização já foi confirmada pelo banco de dados, por meio do comando *commit*.

A figura 3.8 mostra um exemplo de como esse tipo de situação é evitada com a suspensão temporária do uso da cache. Duas transações concorrentes ao banco de dados são executadas. Na transação A, a consulta 1 faz o uso habitual da cache. Na transação B, o comando de escrita 5 é usado, afetando a tabela "item".

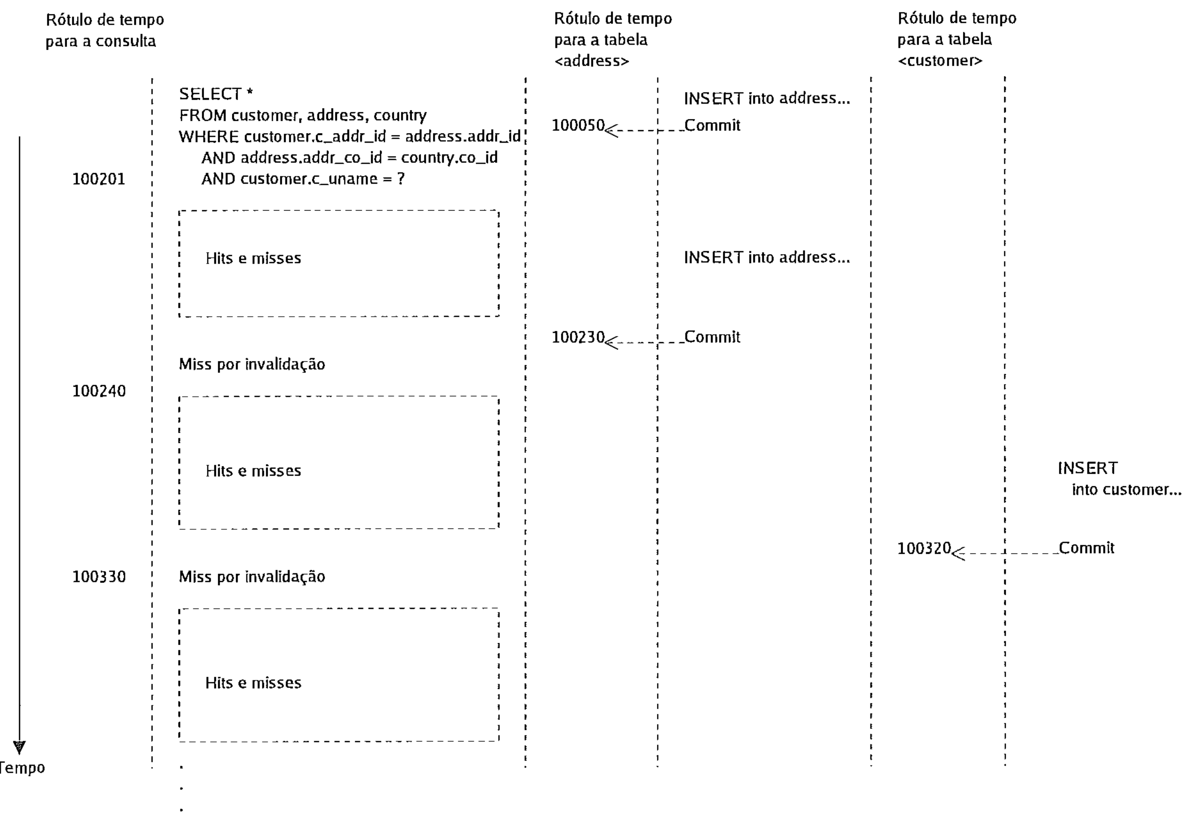

Figura 3.7: Falhas por invalidação

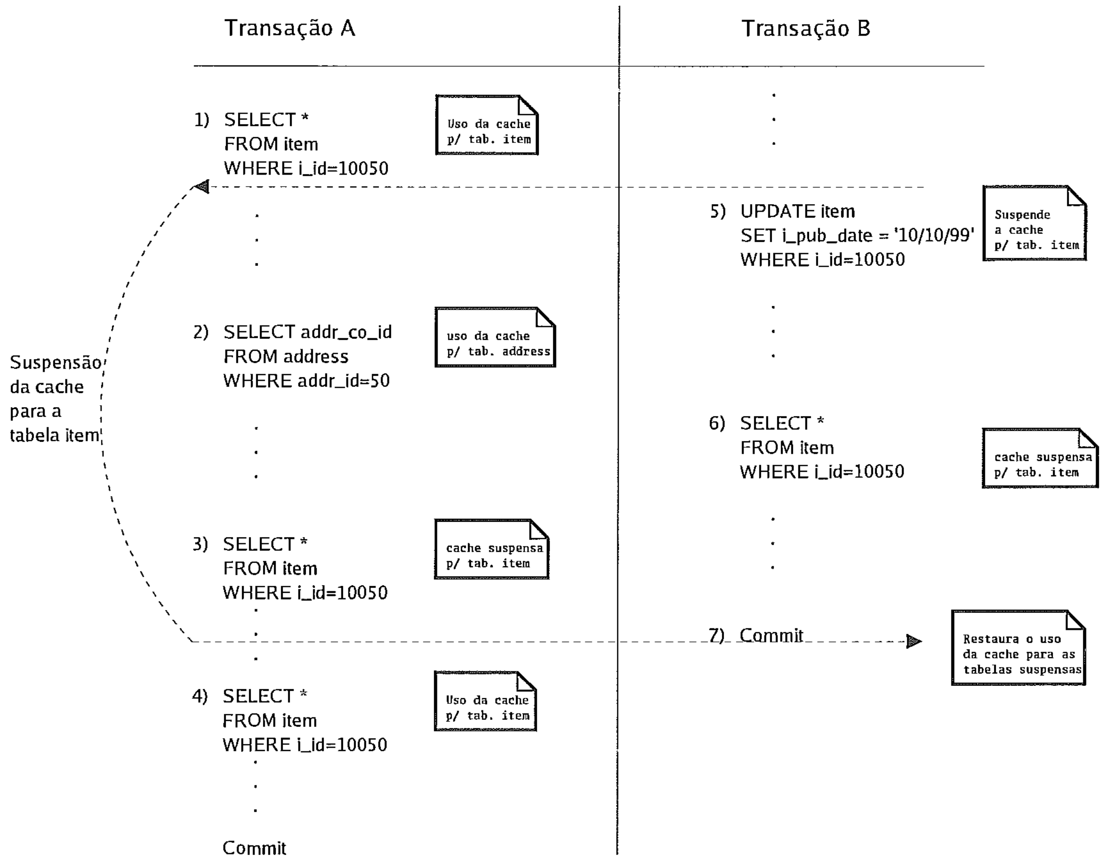

Figura 3.8: Suspensão temporária do uso da cache

A partir deste momento, qualquer consulta em qualquer transação concorrente que faça o uso da tabela "item" é impedida de usar a cache, e é obrigada a procurar o banco de dados, até que a transação B realize o *commit* (7, na figura 3.8). Com isso, evita-se que um valor vindo do banco de dados, como por conseqüência da consulta 6, alimente a cache com um valor não definitivo, sujeito a ser cancelado por um comando *rollback*, e que esse valor possa ser usado com uma outra consulta concorrente, como na consulta 3.

Após o *commit*, é cancelada a suspensão do uso da cache para todas as tabelas usadas em comandos de escrita em uma transação, e as transações concorrentes voltam a poder fazer o uso da cache para essas tabelas, como é o caso da consulta 4. Durante o período de suspensão, consultas que não usem as tabelas suspensas podem fazer o uso habitual da cache, como é o caso da consulta 2.

O controle das tabelas que acarretam suspensão é feito por cada conexão ao banco de dados. Cada transação usa uma conexão diferente, e por isso a cada tabela é associada uma coleção de conexões cujas transações têm alterações em curso na tabela.

Durante uma transação, quando ocorre um comando UPDATE, INSERT ou DELETE, a conexão associada à transação é adicionada à coleção de conexões de cada tabela envolvida no comando.

Durante uma consulta, para cada tabela envolvida desta vez na consulta, é verificado se a coleção de conexões é um conjunto vazio. Caso o seja para todas as tabelas envolvidas na consulta, a cache está liberada para uso. Caso haja pelo menos uma conexão associada a qualquer dessas tabelas, o uso da cache é suspenso temporariamente.

Quando uma transação realiza *commit* ou *rollback*, a conexão associada à transação é retirada das coleções de conexões das tabelas que sofreram alteração. Dessa forma, caso não haja outras conexões nessas coleções, fica liberado o acesso a cache que envolva essas tabelas.

A suspensão do uso da cache pode causar falhas em cache, mas diferente das invalidações, não tem o efeito indesejável de eliminar tuplas da cache.

3.4 Limitações do *driver*

A seguir são descritas algumas das limitações atuais do *driver* dCache.

Stored Procedures e funções

As *stored procedures* são rotinas de comandos em SQL que são compilados e executados no servidor. Por serem executados externamente ao *driver*, sem que os comandos SQL envolvidos sejam revelados ao *driver*, não há manutenção de consistência de cache entre as *stored procedures* e os comandos SQL que passam pelo dCache.

Uma solução possível para manter a consistência nesse caso seria submeter o código fonte das *stored procedures* a um tradutor que inserisse nesse código fonte comandos de invalidação remota da cache. Seria criada assim uma segunda versão da *stored procedure* que pudesse ser executada pelo *driver* sempre que a primeira fosse solicitada.

O *driver* poderia se encarregar de fazer essa tradução de forma transparente para alguns SGBDs, obtendo o código fonte da *stored procedure,* por meio de comandos que revelam o código fonte de stored procedures, como o comando "SHOW CREATE PROCEDURE" do MySQL, e recriando-as com comandos como "CREATE PROCEDURE".

Essa abordagem seria porém bastante dependende de cada SGBD, pois cada sistema gerenciador de banco de dados tem a sua própria sintaxe para stored procedures. Seria portanto necessário implementar módulos tradutores específicos para cada SGBD.

Triggers

O *driver* dCache não mantém consistência em relação a operações realizadas por triggers no SGBD, que também são procedimentos executados no servidor de banco de dados, em resposta a eventos ocorridos no banco de dados.

Alterações ao banco de dados externas

Não há garantia de consistência dos dados em cache se forem feitas alterações no banco de dados que sejam externas, ou seja, que não utilizem o *driver* dCache. Caso alguma alteração assim precise ser feita, os dados em cache do dCache precisam ser removidos.

Subqueries

A execução de *subqueries*, ou consultas encadeadas, não foi implentada para o *driver*.

Invalidações devido a UPDATE

As invalidações no dCache são causadas pelos comandos INSERT e DELETE. Não são tratadas invalidações que poderiam ocorrer devido ao uso do comando UPDATE. Por exemplo, o comando:

```
UPDATE item
SET i_related1 = 525
WHERE i_id = 840
```

deveria acarretar a invalidação das tuplas de uma relação resultante da consulta:

```
SELECT *
FROM item
WHERE i_related1 = 524
```

caso a relação contivesse, antes do uso do comando UPDATE, uma tupla com os atributos i_related = 524 e i_id = 840.

Isso ocorre porque o modelo de invalidação seletiva escolhido para o comando UPDATE, que envolve a análise dos predicados das consultas, ainda não foi implementado. Uma próxima versão do dCache analisará o predicado das consultas para determinar se a relação deve ser invalidada ou não, com base nos atributos alterados. Invalidando apenas a relação, evita-se a invalidação de todas as tuplas do mesmo tipo de consulta, como ocorre nas invalidações devido a INSERT e DELETE.

Níveis de isolamento das transações

O nível de isolamento *read commited* (nível 1 do padrão ANSI SQL-92) é obtido por meio da suspensão da cache. O nível de isolamento read uncommited (nível 0) pode ser obtido desligando-se a suspensão entre as transações, mas mantendo-a para a mesma transação. Os demais níveis, *repeatable read* (nível 2) e *serializable* (nível 3), não foram implementados.

O *driver* repassa para o SGBD o nível escolhido pela aplicação, e por isso, para os níveis que não foram implementados, só é garantido o isolamento entre transações que ocorram usando o banco de dados, e não para transações que usam a cache. Por isso o uso para os níveis que não foram implementados é impraticável.

Capítulo 4

Avaliação experimental do dCache

Este capítulo descreve os experimentos realizados para avaliar o *driver* dCache no contexto de um cluster Web executando um serviço de comércio eletrônico. Uma implementação do benchmark TPC-W foi utilizada para esse propósito.

Os experimentos foram estruturados na tentativa de se observar os seguintes efeitos:

- Benefício da cache na vazão total de um servidor dependendo do número de operações de invalidação e atualização da cache, conforme é variada a taxa operações de escrita no banco de dados.
- Benefício da cache em função da mudança de um fator de escala. Variou-se o tamanho da tabela item, em experimentos com 100 mil e 1 milhão de registros[1].
- Desempenho da cache para os diversos tipos de consultas
- Uso de memória pela cache entre os diversos tipos de consultas

Para se efetuar a variação da taxa de operações de escrita, utilizaram-se os três tipos de carga do TPC-W, "Browsing Mix", "Shopping Mix" e "OLTP Mix", que diferem, entre outros aspectos, pela proporção de interações que envolvem escritas no banco de dados, conforme ressaltado antes na tabela 2.2.

Observou-se o efeito das invalidações, mas constatou-se que o efeito de ocupação da CPU do SGBD é mais importante para o desempenho da cache. A diferença na proporção de consultas entre as três cargas, que acarreta uma ocorrência maior de consultas mais complexas para a carga "Browsing Mix" do que para a carga

[1]Em ambos os casos, o armazenamento total do banco de dados em disco ocupou aproximadamente 1GB, conforme pode ser verificado no apêndice C, tabela C.1.

"Shopping Mix" e um número maior para esta carga do que para a carga "OLTP Mix", tem o efeito de ocupar o servidor de banco de dados de forma diferenciada, e de beneficiar mais o uso da cache para os casos em que o SGBD está mais ocupado.

Observou-se também que no fator de escala de 1 milhão de itens o uso da cache tem melhores resultados do que no fator de escala de 100 mil itens, porque o custo maior das consultas no fator 1 milhão beneficia o uso da cache.

Também foi observado que os diferentes tipos de consultas causam ocupações diferentes em memória na cache, devido a fatores como a freqüência de uso da consulta, o número de invalidações sofridos por cada tipo de consulta, e a distribuição estatística dos parâmetros que ocorrem para cada tipo de consulta, que pode seguir distribuição uniforme ou de cauda pesada.

4.1 Ambiente experimental

O ambiente de hardware utilizado nos experimentos tem a sua arquitetura ilustrada na figura 4.1. Trata-se de 4 nós de um cluster com sistema operacional Linux, kernel 2.6.7, cada um usando um processador Intel Pentium 4, de 2,8 GHz, e 1GB de memória RAM, interconectados por um switch gigabit ethernet, e utilizados da seguinte maneira:

- um nó para a emulação de navegadores, executando em um runtime Java, Standard Edition, versão 1.4.2
- um para o servidor Web Apache, versão 2.0.46
- um para o servidor de aplicação Apache Tomcat versão 5.0.28, executando em um *runtime* Java, Standard Edition, versão 1.5.0. O servidor de aplicação, a aplicação e o driver dCache compartilham o mesmo *runtime* java neste nó.
- um para o servidor de banco de dados MySQL, versão 4.1.4, usando o motor de armazenamento "MyISAM".

A comunicação entre a aplicação e o SGDB foi realizada pelo *driver* dCache em associação com o *driver* JDBC "MySQL connector", versão 3.0.15. A comunicação entre o servidor Web Apache e o servidor de aplicação Tomcat foi feita pelo módulo mod_jk2 para o Apache, utilizando o protocolo AJPv1.3. Os dados de ocupação de CPU nos nós foram coletados com o Sysstat, v.4.0.7 [45].

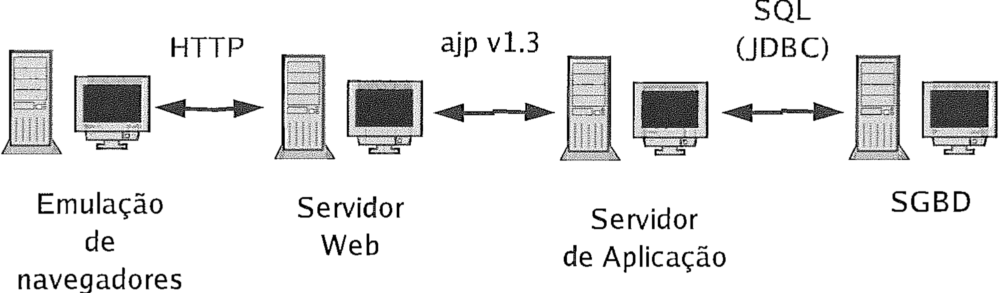

Figura 4.1: Arquitetura do ambiente de hardware utilizado nos experimentos

4.2 Benchmark utilizado

A implementação do benchmark TPC-W utilizada deriva da que foi inicialmente desenvolvida na Universidade de Wisconsin-Madison, em 1999 [46]. Trata-se de uma versão não oficial baseada na especificação 1.0, implementada para o SGDB IBM DB2. Posteriormente essa versão foi adaptada para o banco MySQL e distribuída pelo consórcio ObjectWeb [47], que promove o desenvolvimento e teste de software *middleware* e de fonte aberto. A implementação efetivamente utilizada adaptamos a partir desta última, e as adaptações mais relevantes foram as seguintes:

1. Os dados do carrinho de compras, antes de a compra ser efetuada, eram armazenados em tabelas no banco de dados. Isso acarreta uma ocupação desnecessária do banco de dados, uma vez que não há exigência de persistência para esse tipo de dados. A especificação do TPC-W não obriga o armazenamento desses dados em tabelas [38], e por isso modificou-se a implementação para tratar esses dados como dados de sessão.

2. A procura por palavras chave, normalmente realizada usando-se o operador "like" em SQL, mostrou acarretar problemas sérios de desempenho no MySQL quando a tabela em questão tinha muitos registros. Isso porque este SGBD precisa realizar a varredura completa da tabela para encontrar registros quando se trata de partes de palavras que podem ocorrer não necessariamente no início de uma frase ou cadeia, uma vez que quando se usa esse operador no MySQL são utilizados índices para as cadeias completas, mas não para as palavras que as compõem. O operador foi substituído pelo operador "match...against", que no MySQL permite a utilização associada a um índice de palavras para busca textual, o que tornou a busca algumas ordens de grandeza mais rápida em alguns casos, e a comparação com o uso

da cache mais justa. A solução para esse problema adotada na distribuição do TPC-W antes da adaptação [47] era baseada na função de comparação fonética "soundex", que não produz resultados exatos e por isso a rejeitamos.

3. Descobrimos que uma função do código fonte encarregada de gerar números aleatórios para as páginas que contém ofertas promocionais gerava números aleatórios apenas para a faixa de fator de escala de 10.000 itens. Não há qualquer indicação, na documentação que acompanha a distribuição, de que essa função precise ser modificada para experimentos com outros fatores de escala, como de fato precisa, para estar de acordo com a cláusula 2.2.18 da especificação do TPC-W v.1.8 [38]. Em experimentos anteriores, esse descuido dos implementadores do benchmark havia comprometido gravemente os resultados, beneficiando excessivamente o uso da cache, uma vez que detectou-se uma altíssima taxa de acertos em cache para experimentos com fator de escala de 100.000 e 1.000.000 de itens. É possível que outros trabalhos baseados nessa distribuição do benchmark, principalmente os que avaliam o uso de cache, estejam com resultados comprometidos. Uma correção foi aplicada para que os números aleatórios sejam gerados dentro das faixas corretas.

Apesar das adaptações, a implementação ainda se desvia de uma implementação oficial do TPC-W pelo menos nos seguintes aspectos:

- Não existe comunicação com um emulador de gateway de pagamento, que serviria para simular a operação de solicitação de confirmação de pagamento com a administradora de cartão de crédito, no momento da confirmação da compra.
- Não são feitas as conexões via SSL ou TSL para as transações seguras.
- O programa que faz a criação e população da base de dados não gera os nomes de autores e títulos dos livros seguindo a especificação. Uma ferramenta chamada "wgen" fornecida pelo próprio Transaction Processing Performance Council foi utilizada para esse propósito.

Esses desvios da especificação oficial podem ter os seguintes impactos nos resultados:

- Em conexões com criptografia para as transações seguras as transferências ocupam mais a CPU do servidor Web, mas não do servidor de aplicação ou do servidor de banco de dados, onde a cache tem influência direta. Haveria

um peso maior para a carga do servidor Web, talvez diminuindo um pouco a importância da cache na vazão total. Entretanto, essas conexões, que deveriam ser feitas para as interações "Buy request", "Buy confirm", "Order inquiry" e "Order display", totalizam apenas 1,99% das interações da carga "Browsing Mix", 5,21% das interações da carga "Shopping Mix" e 23,38% das interações da carga "OLTP Mix". Ou seja, o efeito teria menos importância nas cargas em que se obteve maior ganho. Em uma conexão segura, até mesmo as imagens são criptografadas, mas para essas quatro interações as páginas têm apenas 2 a 3 imagens entre 1KB a 2KB.

- O acesso ao emulador de gateway de pagamento é feito pelo servidor de aplicação e também envolve criptografia. A criptografia é assimétrica e a chave privada usada deve ter no mínimo 1.024 bits. Haveria uma carga maior de processamento justamente onde a cache é manipulada - no servidor de aplicação, potencialmente prejudicando o uso da cache. Entretanto, o emulador de gateway de pagamento é acessado apenas durante a interação "Buy Confirm", que ocorre em 0,69% das interações na carga "Browsing", 1,20% na carga "Shopping" e 10,18% na carga "OLTP". Mais uma vez, o efeito teria menos importância nas cargas em que se obteve maior ganho.

4.3 Descrição dos experimentos

São realizadas duas séries de experimentos. Na primeira série, é usado o fator de escala de 100 mil itens e a carga é gerada por 300 navegadores emulados. Na segunda série, é usado o fator de escala de 1 milhão de itens e a carga é gerada por 50 navegadores emulados. Em cada série, são feitos experimentos com as três cargas descritas na tabela 2.3, ou seja, Browsing Mix, Shopping Mix e OLTP (ou "Ordering") Mix. Os tamanhos das tabelas usadas nas duas séries é mostrado na tabela C.1, do apêndice C.

Para cada uma das cargas e fator de escala, é utilizado o *driver* dCache de duas formas em experimentos distintos:

1. Com cache: tamanho máximo de cache de parâmetros de 1 milhão de entradas e tamanho máximo de cache de tuplas (RowHash) também de 1 milhão de entradas. Esse tamanho é mais que suficiente para manter os dados sem que haja necessidade de substituição dos dados em cache.

2. Sem cache: neste modo, o *driver* não usa cache e limita-se a usar diretamente o *driver* MySQL Connector para todas as operações de acesso ao banco de

dados, e a registrar dados segundo a sua instrumentação. Não há instrumentação na aplicação, mas somente no *driver*. Por isso foi utilizada esta versão do *driver* sem cache, em vez de apenas o driver do MySQL.

Cada medida é tomada durante 30 minutos, e é feita após um período de aquecimento de no mínimo 30 minutos. Durante o período de medida, são aferidas as taxas de ocupação de CPU dos nós com o servidor de banco de dados, com o servidor de aplicação e com o servidor Web. Ao final de cada período, são recolhidos os dados de instrumentação do *driver*. Entre um experimento e outro, os servidores de aplicação e o SGBD são reiniciados.

Devido ao fato de essa implementação do benchmark apresentar resultados em uma faixa de interações por segundo (WIPS) baixa, optou-se por apresentar os resultados em interações por minuto (WIPM).

4.3.1 Instrumentação do *driver*

A instrumentação do *driver* registra uma série de informações durante a execução do *driver* que podem ser consultadas a qualquer momento por meio de um serviço, que responde a requisições em uma porta escolhida. No apêndice C encontram-se essas informações recolhidas ao final da realização de cada experimento envolvendo a cache. Os tipos de informações, que estão disponíveis para cada estrutura de consulta SQL separadamente, são as seguintes:

Para as operações de leitura

São medidos a quantidade de ocorrências de acerto (*hit*) e de falha (*miss*), custo total e custo médio dos acertos e falhas, em microssegundos, quantidade de ocorrências de falhas por invalidação e quantidade de ocorrências falhas por suspensão do uso da cache.

O custo de acerto ou falha neste caso significa o tempo que se passa entre o momento em que a aplicação solicita uma referência a um valor de atributo ao *driver* e o momento em que o *driver* o concede à aplicação, sendo que esse tempo é acumulado para todas as solicitações dentro de uma mesma consulta. Se durante uma dessas solicitações o valor em cache não é encontrado, é feito um acesso ao banco de dados, e a consulta passa a ser considerada uma falha. Se nenhuma das solicitações para a mesma consulta envolver acesso ao banco, a consulta é tratada como um acerto.

Para as operações de escrita no banco de dados

São registrados o número de operações de escrita, custo total e custo médio das operações de escrita, em microssegundos, número de operações de atualização dos valores em cache, custo total e médio da atualização de valores em cache e tamanho utilizado para as tabelas de dependência global.

Para todas as operações

São medidos a quantidade de acertos e falhas na cache de comandos SQL (Statement Cache), e o custo total e médio desses acertos e falhas, em microssegundos.

Uma falha na cache de comandos SQL significa a necessidade de se fazer o parsing do comando, e o custo médio dessas falhas revela o tempo aproximado necessário para o parsing.

4.4 Primeira série

Na primeira série, realizada com fator de escala de 100 mil itens e carga gerada por 300 navegadores emulados, cada experimento sem cache foi precedido de um período de aquecimento de 30 minutos, e cada experimento com cache por um período aquecimento de 60 minutos. Não houve diferença apreciável do desempenho em função do período de aquecimento em experimentos sem cache realizados previamente, e por isso esse período foi fixado em 30 minutos para os experimentos das duas séries.

4.4.1 Desempenho

A figura 4.2 mostra os resultados das medidas de interações web por minuto. Há um ganho de desempenho nessas medidas com o uso da cache, de 55% para a carga "Browsing Mix" e de 17% para a carga "Shopping Mix". Entretanto há uma redução de desempenho para a carga "OLTP Mix", de 14%.

Vazão e ocupação de CPU

Existe uma correspondência entre o ganho de desempenho em vazão no uso da cache e a taxa de ocupação de CPU do servidor de banco de dados. Esse resultado era naturalmente esperado. Na figura 4.3(a) verifica-se que o ganho da cache é tanto maior quanto maior é a taxa de uso de CPU no SGBD, ou seja, quanto mais o SGBD se caracteriza como gargalo de desempenho. Existe um custo adicional

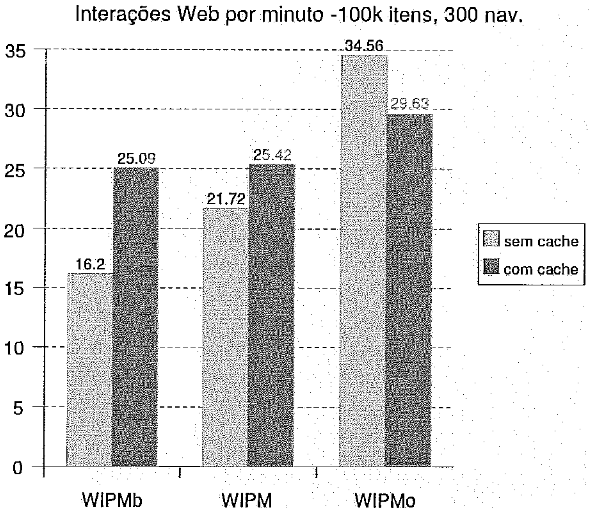

Figura 4.2: Interações Web por minuto, com e sem cache, usando-se as cargas Browsing (WIPMb), Shopping (WIPM) e OLTP (WIPMo) - primeira série

considerável do tratamento e operação da cache pelo *driver*, que ocorre no servidor de aplicação, conforme pode ser verificado na figura 4.3(b), e isso contribui para explicar a degradação de desempenho verificada para a carga "OLTP Mix", em que o banco de dados não era o gargalo.

Esse custo adicional ocorre no servidor de aplicação, onde o *driver* é executado, e o processamento adicional da cache compete pela CPU desse servidor com outras tarefas do servidor envolvendo a lógica da aplicação e geração dinâmica das páginas. Essa competição não existe quando os dados são advindos do SGBD. Por isso a diferença notada na figura 4.3(b) entre o uso de CPU no servidor de aplicação com e sem cache é tão acentuada.

Não há diferença apreciável de uso de CPU no servidor web, conforme a figura 4.3(c), por não ser ele beneficiado ou prejudicado com o uso de cache de banco de dados.

Espera por dados do SGBD

A análise do tempo de espera, pelo *driver*, por dados do SGBD confirma a hipótese da influência da ocupação do banco de dados no desempenho da cache. A figura 4.4(a) mostra o tempo total gasto na espera pelo banco de dados, dividido pelo número de navegadores, para as três cargas, sem o uso de cache. Esse tempo é dividido ainda em leituras e escritas, e pode ser conferido numericamente na tabela 4.1. O tempo de espera para a carga "Browsing Mix" é muito maior que o tempo

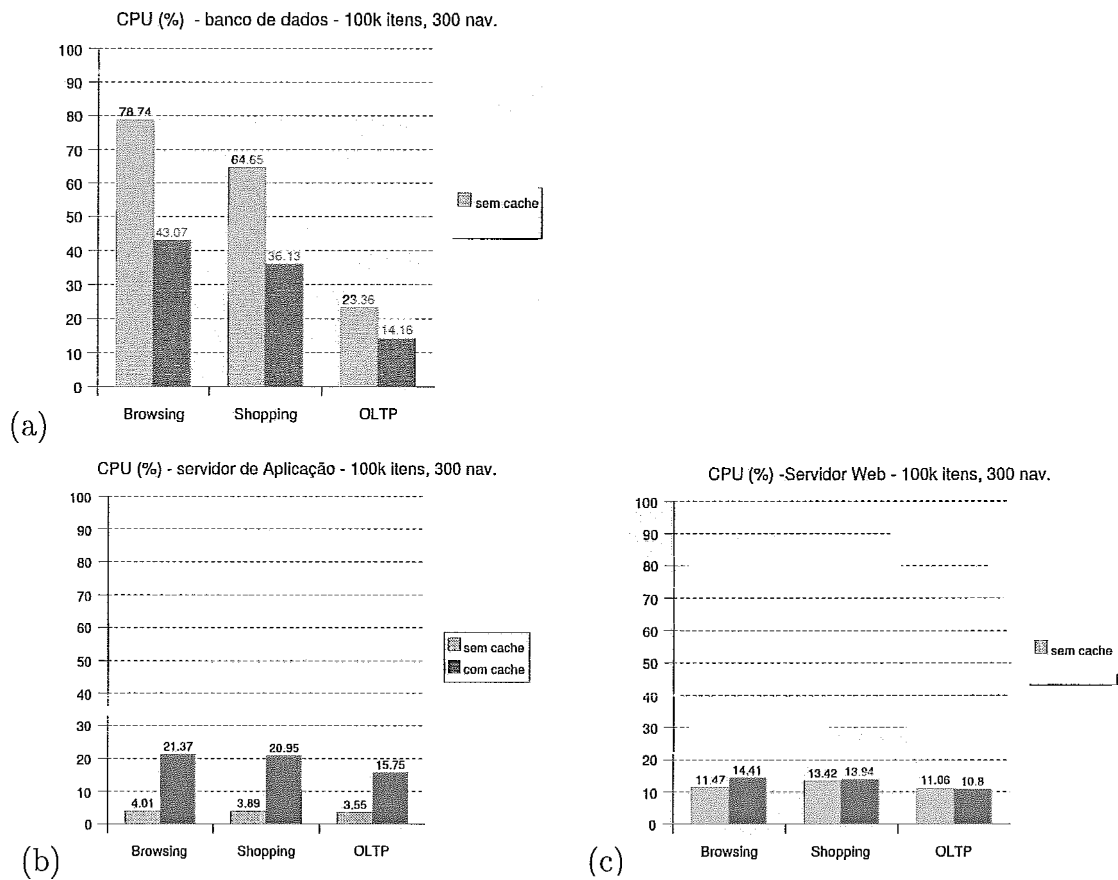

Figura 4.3: Taxa de ocupação da CPU das máquinas com o servidor de banco de dados (a), servidor de aplicação (b) e servidor Web (c) - primeira série

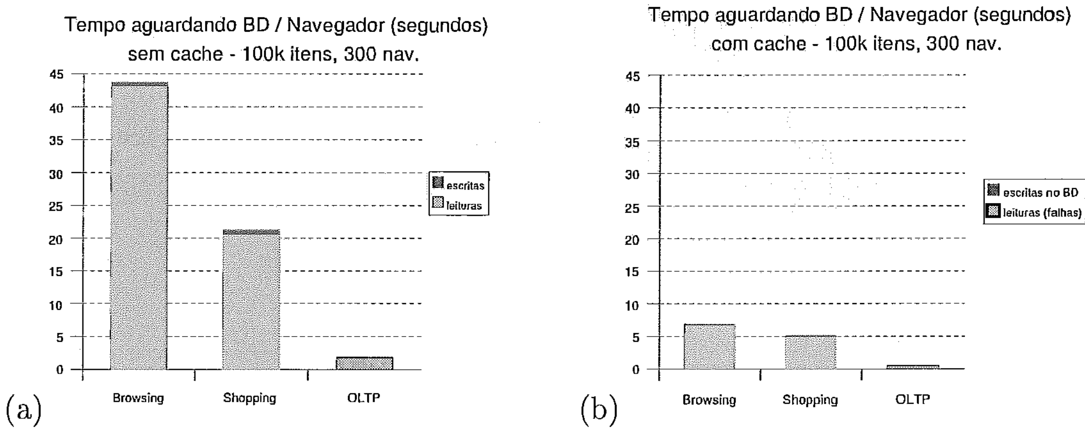

Figura 4.4: Tempo aguardando banco de dados, em segundos, por navegador, (a) sem cache e (b) com cache - primeira série

de espera para a carga "OLTP Mix", e significativamente maior do que a carga "Shopping Mix". O uso da cache resulta, conforme a figura 4.4(b), em uma redução expressiva do tempo de espera por dados vindos do SGBD para as três cargas. E na comparação da redução para as três cargas, a redução é tanto maior quanto maior o tempo de espera, ou seja, quanto maior a ocupação do SGBD. É interessante também notar por essas figuras que as operações de leitura em todos os casos têm custo muito maior que as operações de escrita no banco de dados.

A redução substancial no tempo de espera por dados do banco de dados, na figura 4.4, não se traduz em aumento substancial de vazão total do servidor, conforme a figura 4.2, porque o acesso a banco de dados, entre o servidor de aplicação e o SGBD, é apenas um dos fatores que influenciam na vazão, cujo cálculo é feito conforme foi descrito em 2.2.1.1. Existe uma série de outros custos fixos, como a busca ao servidor web de imagens da página, e outros que podem até aumentar com o uso de cache devido à competição de processamento entre linhas de execução envolvendo o *driver* e a aplicação no servidor de aplicação, como a geração dinâmica da página. Esses custos também têm influência no resultado final da vazão, e de acordo com a lei de Amdahl [29], o ganho de desempenho que pode ser obtido usando-se um modo mais rápido de execução é limitado à fração de tempo em que esse modo mais rápido pode ser usado. No caso, esse modo mais rápido é o acesso ao banco de dados.

Taxas de acertos e falhas em cache

A taxa de acertos global nos experimentos com cache para as três cargas é revelada na figura 4.5(a). Como esperado, existe uma tendência de maior taxa de acertos

		Browsing	Shopping	OLTP
Sem cache	leituras	43.20	20.62	1.76
	escritas	0.54	0.68	0.16
	total	43.74	21.29	1.92
Com cache	leituras (falhas)	6.78	4.99	0.52
	escritas	0.15	0.22	0.09
	total	6.93	5.21	0.60

Tabela 4.1: Tempo aguardando banco de dados, em segundos, por navegador - primeira série

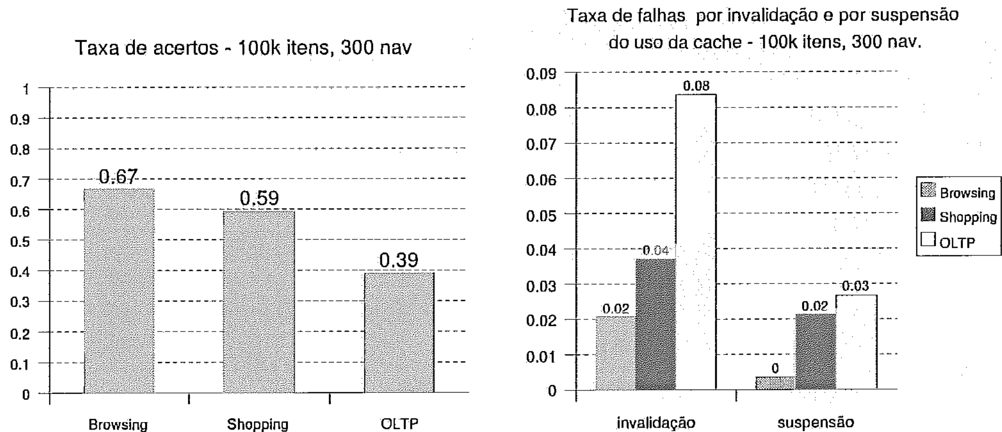

Figura 4.5: Taxa de acertos (a) e taxa de falhas por invalidação e por suspensão do uso da cache (b) - primeira série

quanto menor a taxa de escritas, que pode ser verificada na comparação da tabela com a proporção de leituras e escritas das três cargas da tabela 2.2, mostrada durante a descrição do TPC-W feita no capítulo 2.

A taxa de falhas que ocorrem por invalidação e a taxa de falhas que ocorrem devido a suspensão do uso da cache são mostradas na figura 4.5(b). Essas taxas também acompanham a proporção citada de leituras e escritas das interações. Quanto maior a quantidade de escritas, maior a taxa de falhas por invalidação, e maior a taxa de falhas por suspensão do uso da cache, devido ao tempo maior em que as consultas concorrem com transações que envolvem escritas. Mesmo taxas de falha por invalidação pequenas como essas, que estão todas abaixo de 10%, podem reduzir significativamente a taxa de acertos, pois a granularidade das invalidações é baixa, já que uma invalidação causa a eliminação de toda a cache de tuplas (RowHash) associada a um determinado tipo de consulta.

Peso de consultas complexas

Os diferentes tipos de consulta ao banco de dados têm diferentes pesos sobre o SGBD. Há consultas que envolvem a busca de uma única tupla, indexada por um atributo. Há outras que envolvem a busca de várias tuplas resultantes da junção de tabelas, há também aquelas que retornam informações segundo critérios de ordenação e agregação de informações, como por exemplo a operação de adição de diversos registros, e há ainda as que envolvem combinações dessas operações. Os tipos de consulta que ocorrem no benchmark podem ser verificados no apêndice C, nas figuras C.1, C.2 e C.3.

As três cargas têm proporções diferentes de ocorrência desses diferentes tipos de consulta. No apêndice C, a tabela C.4 revela informações detalhadas coletadas após o experimento com a carga "Browsing", com uso de cache. É possível verificar, dentre outras informações, o custo total de falhas de cada tipo de consulta separadamente. O mesmo tipo de informação está disponível para os experimentos com as cargas "Shopping" (tabela C.5) e "OLTP" (tabela C.6).

As diferenças nas proporções das interações entre as três cargas ajudam a explicar a variação da ocupação do SGBD. A figura 4.6(b) mostra o fracionamento da ocorrência das interações para as três cargas. As interações "Best Sellers" (BestSell) e "New Products" (NewProd) ocorrem juntas em 22% das interações da carga "Browsing", em 10% das interações de "Shopping" e em apenas 0,92% das interações de "OLTP". Entretanto, a consulta S6, que é realizada na interação "Best Sellers", é sozinha responsável por 69% do custo de falhas na carga "Browsing", 62% do custo de falhas na carga "Shopping" e 18% do custo de falhas na carga "OLTP", conforme mostra a figura 4.6(a). O custo desse tipo de consulta, que envolve junções, agregações, somas e ordenações, tem impacto decisivo na ocupação do SGBD.

Invalidações e taxa de acertos

A tabela 4.2 mostra a taxa de acertos, número de invalidações, número de acessos (acertos + falhas) e a taxa de falhas ocorridos por invalidação, para as consultas S6 e S15. Mesmo uma taxa pequena de falhas por invalidação pode ter impacto significativo na taxa de acertos para essas consultas. Isso ocorre porque essas consultas são parametrizadas, e uma invalidação causa a eliminação da cache de todas as tuplas para o tipo de consulta em questão. É preciso um tempo para que a cache acumule tuplas para vários parâmetros, após uma invalidação. E se a invalidação é muito freqüente, esse tempo não é suficiente para garantir uma taxa de acertos satisfatória.

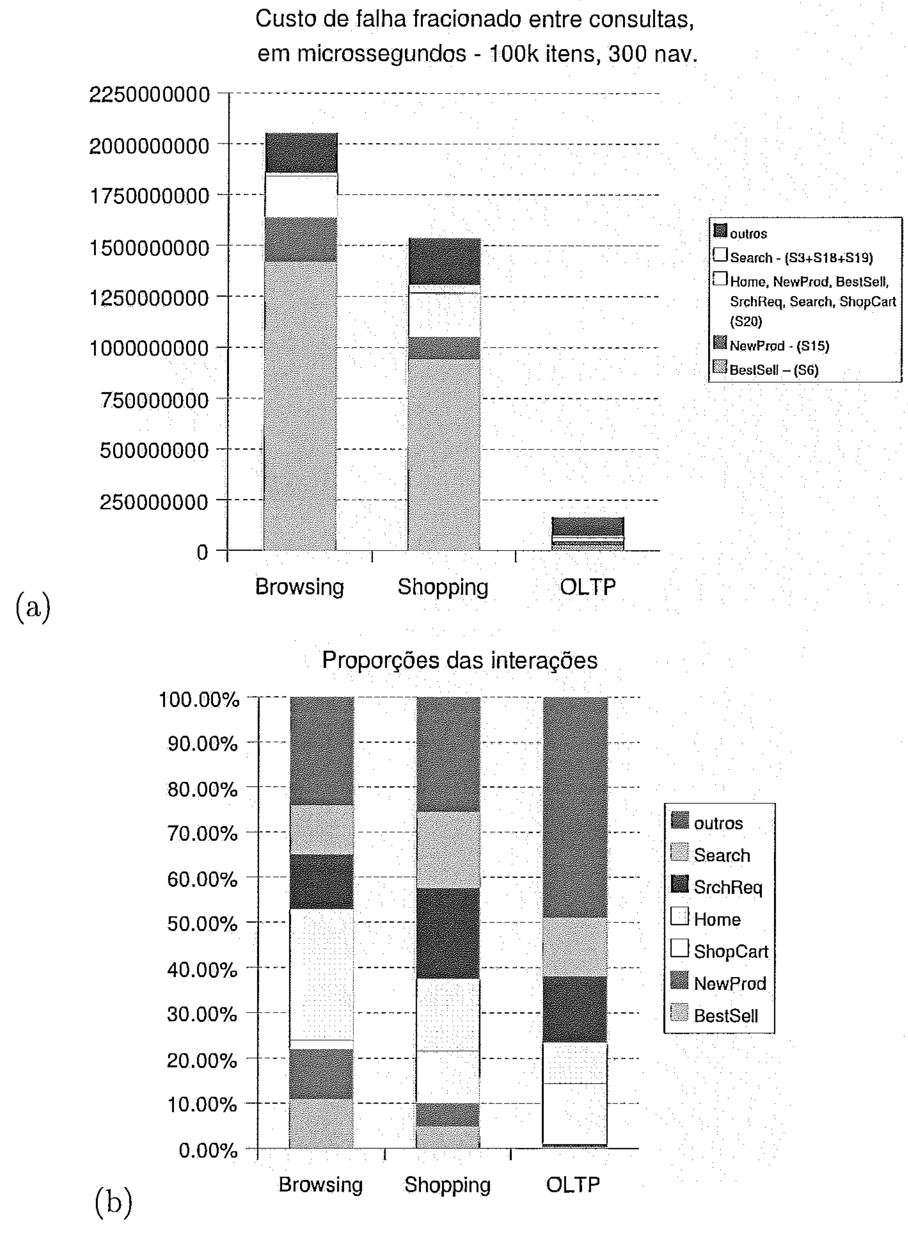

Figura 4.6: Custo de falhas fracionado entre interações, em microssegundos (a) e fracionamento da ocorrência das interações - primeira série

Consulta S6: SELECT *, SUM(ol_qty) as sum FROM item, author, order_line WHERE item.i_id = order_line.ol_i_id AND item.i_a_id = author.a_id AND order_line.ol_o_id > ? AND item.i_subject = ? GROUP BY i_id, i_title, a_fname, a_lname ORDER BY sum DESC LIMIT 50

Carga (S6)	Taxa de acertos	invalidações	acessos	Taxa de falhas por invalidação
Browsing	41%	334	6.258	5%
Shopping	16%	481	2.913	17%
Ordering	2%	226	276	82%

Consulta S15: SELECT i_id, i_title, a_fname, a_lname FROM item, author WHERE item.i_a_id = author.a_id AND item.i_subject = ? ORDER BY item.i_pub_date DESC,item.i_title LIMIT 0,50

Carga (S15)	Taxa de acertos	invalidações	acessos	Taxa de falhas por invalidação
Browsing	81%	55	6.156	0,9%
Shopping	74%	39	2.983	1%
Ordering	24%	50	316	16%

Tabela 4.2: Taxa de acertos, invalidações e taxa de falhas por invalidação - consultas S6 e S15

4.4.2 Uso de memória

Uma questão surpreendentemente não trivial no que diz respeito a programas em Java é saber com precisão a memória ocupada por objetos com estruturas de dados complexas alocadas na *heap* da máquina virtual Java. É possível entretanto saber o número médio de bytes ocupado por objetos das classes básicas da API do Java, para uma dada máquina virtual [48]. Com base nesses valores médios, foi possível determinar que a ocupação aproximada dos objetos em cache nos experimentos foi da ordem das dezenas de megabytes, em contraposição ao tamanho das bases de dados no SGBD, que ocupavam cerca de 1GB em disco.

Conforme visto anteriormente, existem três tipos de estrutura de cache LRU no dCache. A "Statement Cache", que armazena estruturas de comandos SQL que já sofreram parsing, a cache de parâmetros, ou "Param Cache", que relaciona consultas e seus parâmetros a relações, e a cache de tuplas, ou "Row Hash", que relaciona índices das tuplas nas relações a tuplas únicas em memória. Além dessas estruturas LRU, há as tabelas de dependência global, permanentes, que contêm o último valor (mais atualizado) de atributos que sofreram atualização.

A "Statement Cache", ou cache de comandos SQL, tem tamanho pequeno, necessitando de não mais que algumas dezenas de entradas, pois precisa acomodar apenas as estruturas dos comandos SQL, que dentro de uma aplicação são poucos.

Carga	Browsing	Shopping	OLTP
Tam. total máximo da cache de tuplas	419.270	432.893	396.902
Tamanho da cache de parâmetros	111.841	122.452	132.687
Tamanho das tabelas de dep. global	3.771	7.517	19.447

Tabela 4.3: Entradas da cache de tuplas, de parâmetros e das tabelas de dependência global usadas na primeira série de experimentos

Nos experimentos, são apenas as consultas de S1 a S26, e os comandos de escrita no banco de dados I1 a I5 e U1 a U3, descritos na tabela C.4. A consulta S25 é uma exceção, pois trata-se de diversas strings de comandos SQL diferentes, que foram geradas pela aplicação por concatenação, contendo cada qual um nome de uma tabela temporária diferente. Por conveniência, essas consultas foram agrupadas sob o mesmo identificador S25. O tamanho máximo fixado para a "Statement Cache" para os experimentos foi de 1000 entradas.

A cache de parâmetros, que teve seu tamanho máximo fixado em 1 milhão de entradas, teve após a primeira série de experimentos os números de entradas usadas mostrados na tabela 4.3. A mesma tabela também mostra os tamanhos utilizados das tabelas de dependência global e a soma dos tamanhos máximos das caches de tuplas. A variação acentuada do tamanho, ou número de entradas, das tabelas de dependência global é conseqüência direta da variação da taxa de escritas entre as cargas "Browsing", "Shopping" e "OLTP".

As caches de tuplas, que tiveram também seus tamanhos máximos limitados a 1 milhão de entradas, apresentaram na realidade tamanhos máximos de utilização diferentes para cada tipo de consulta, conforme pode ser verificado nas tabelas C.4, C.5 e C.6. Alguns fatores influenciam o número máximo de entradas da cache de tuplas utilizadas por um tipo de consulta:

- A freqüência de uso de um determinado tipo de consulta.
- O número de tuplas retornadas por um tipo de consulta.
- A limitação, a algumas poucas possibilidades, do domínio de parâmetros com os quais algumas consultas são feitas, como é o caso de S3 e S15, cujos parâmetros associados estão limitados a uma lista de 24 nomes de assuntos possíveis de livros.
- A limitação do domínio de parâmetros causada pela ocorrência estatística deles, como é o caso das consultas S18 e S19, que realizam a procura de

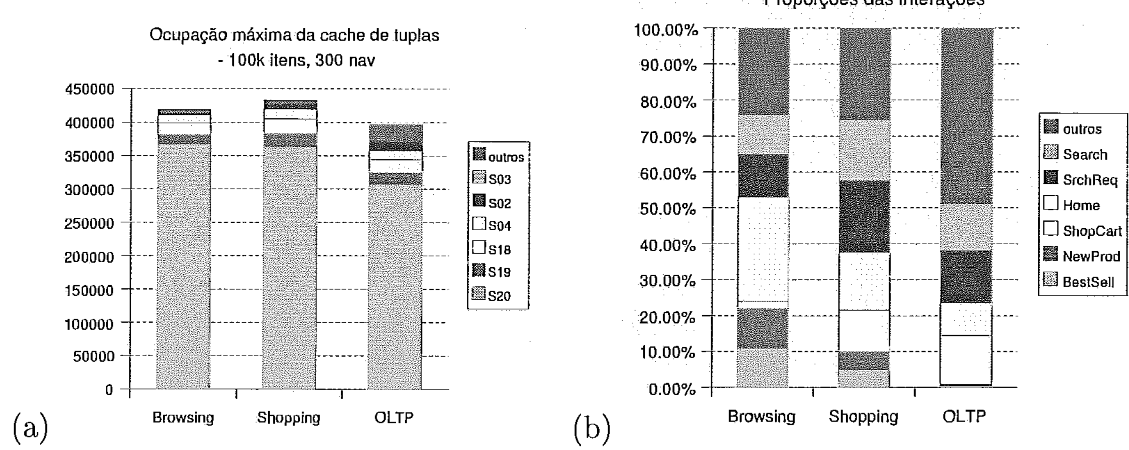

Figura 4.7: Ocupação das caches de tuplas para as três cargas (a), na primeira série de experimentos e as proporções de ocorrência de cada interação.

livros por palavras-chave pelo usuário, e que são utilizadas em conjunto com parâmetros cuja ocorrência segue uma distribuição de cauda pesada.

- A existência de um domínio de parâmetros possíveis não limitado ou muito grande, como é o caso da consulta S20, cujos parâmetros são escolhidos aleatoriamente e seguem uma distribuição quase uniforme.

- A taxa alta de invalidação de um tipo de consulta, como é o caso de diversas consultas que tiveram tamanho máximo de número de entradas na cache de tuplas pequeno porque as invalidações eliminavam todas as entradas com freqüência, como por exemplo S1, S7 e S10.

Freqüência de uso de um tipo de consulta e ocupação da cache

A tabela 4.4 mostra, e a figura 4.7(a) ilustra, para as três cargas, o número máximo de entradas das caches de tuplas de 5 tipos de consulta entre as que mais ocuparam entradas em memória para esse tipo de cache. A consulta S20 é a que tem mais presença nas caches de tuplas, ocupando 88% do espaço para a carga "Browsing", 84% do espaço para a carga "Shopping" e 78% para a carga "OLTP". Trata-se da consulta realizada para oferecer ao usuário cinco itens promocionais, de acordo com uma escolha uniformemente aleatória entre os 100 mil itens do catálogo. Essa consulta aparece em 6 interações, "Search Result", "Search Request", "Home", "Shopping Cart", "New Products" e "Best Sellers", que juntas ocorrem em 76%, 75% e 51% das interações web, respectivamente às três cargas, conforme ilustrado na figura 4.7(b).

Consulta	Browsing	Shopping	OLTP	Ocorre nas interações	Tuplas por consulta	bytes por tupla (aprox)
S20	367.575	364.110	307.980	Search, SrchReq, Home, ShopCart, NewProd, BestSell	5	132
S19	14.284	18.867	16.593	Search	0 a 50	1500
S18	16.918	21.940	19.760	Search	0 a 50	1500
S4	12.194	14.456	13.383	Detail, AdmReq, AdmConf	1	1500
S2	1.701	5.247	14.129	ShopCart	1	250
S3	1200	1200	1200	Search	50	1500
outros	6.598	8.273	25.057	-	-	-

Tabela 4.4: Ocupação máxima das caches de tuplas, em número de entradas, para as três cargas de acordo com o tipo de consulta, e a quantidade de tuplas retornadas a cada consulta - primeira série.

Distribuição estatística de parâmetros e ocupação da cache

As consultas do tipo S20, que retornam apenas 5 tuplas invariavelmente, têm suas tuplas ocupando mais espaço em cache não só porque são mais freqüentes que as outras, mas também porque seus parâmetros ocorrem segundo uma distribuição aleatória quase uniforme dentro de um espaço amostral de 100 mil itens.

As consultas do tipo S18 e S19, que são as consultas de itens por autor e título, podem retornar até 50 tuplas por consulta, mas contribuem para ocupação de 7% do total das caches de tuplas. Essas consultas têm uma ocupação da cache de tuplas menos expressiva do que as consultas do tipo S20 não apenas porque são menos freqüentes, mas também porque seus parâmetros ocorrem segundo uma distribuição de cauda pesada, refletindo o comportamento da escolha de termos de pesquisa pelo usuário.

A figura 4.8 mostra o resultado de uma simulação feita para se verificar estatisticamente a ocorrência dos parâmetros usados com as consultas dos tipos S18, S19 e S20. Foram utilizadas funções originais do código fonte do benchmark TPC-W para gerar os parâmetros que ocorreriam em 100 milhões de requisições para esses três tipos de consultas. A figura 4.8(a) mostra o gráfico da distribuição acumulada da freqüência de ocorrência dos parâmetros das consultas do tipo S19, ou seja, strings de procura por título, em função da posição do parâmetro no ranking. Os dois mil parâmetros mais freqüentes ocorrem em aproximadamente 70% das requisições. A figura 4.8(b) mostra a distribuição acumulada para os parâmetros da consulta S18 - consulta por autor. Os mil parâmetros mais freqüentes aparecem em quase 70%

Tipo de consulta	Falhas por invalidação	taxa de acertos	acessos	Tuplas retornadas por consulta	Máximo de entradas na cache de tuplas
S7	350	8%	6.045	1	87
S10	122	16%	6.045	1	232
S17	0	20%	5.723	1	12.401

Tabela 4.5: Efeito das falhas por invalidação no número de entradas das caches de tuplas de três tipos de consulta - OLTP - primeira série

das requisições. Já a figura 4.8(c), que mostra a distribuição para os parâmetros de S20, ofertas promocionais, indica que para se armazenar os parâmetros mais freqüentes que ocorrem em 70% das requisições, são necessárias 70.000 entradas.

Invalidações e ocupação da cache

Por fim, o efeito das invalidações no número máximo de entradas das caches de tuplas pode ser conferido na tabela 4.5. Essa tabela mostra, para três tipos de consulta do experimento com a carga OLTP com freqüências de acesso semelhantes e com a mesma quantidade de tuplas resultantes de cada consulta, o número de falhas ocorridas por invalidação e o número máximo de entradas usadas nas caches de tuplas de cada tipo. O número máximo de entradas é o maior número de entradas registrado na cache de tuplas durante todo o experimento. Quanto maior o número de falhas por invalidação, menor o número máximo de entradas registrado na cache de tuplas, porque a cada invalidação todas as entradas na cache de tuplas são eliminadas para o tipo de consulta em questão, e por isso o número máximo de entradas registrado permanece pequeno.

Para a consulta S17 não ocorrem invalidações e por isso o número máximo de entradas é sempre crescente, atingindo o número máximo de 12.401 entradas. Esse número é maior que o número de acessos, apesar de apenas uma tupla ser retornada por consulta, porque o número de acessos não inclui os ocorridos durante o período de aquecimento da cache, e as caches de tuplas não são eliminadas entre o período de aquecimento e o período de medida. O propósito do aquecimento, evidentemente, é manter os dados em cache.

4.5 Segunda série

Na segunda série, realizada com fator de escala de 1 milhão de itens e carga gerada por 50 navegadores, cada experimento com cache foi precedido de um período de aquecimento de 3 horas, e os experimentos sem cache continuaram sendo precedidos

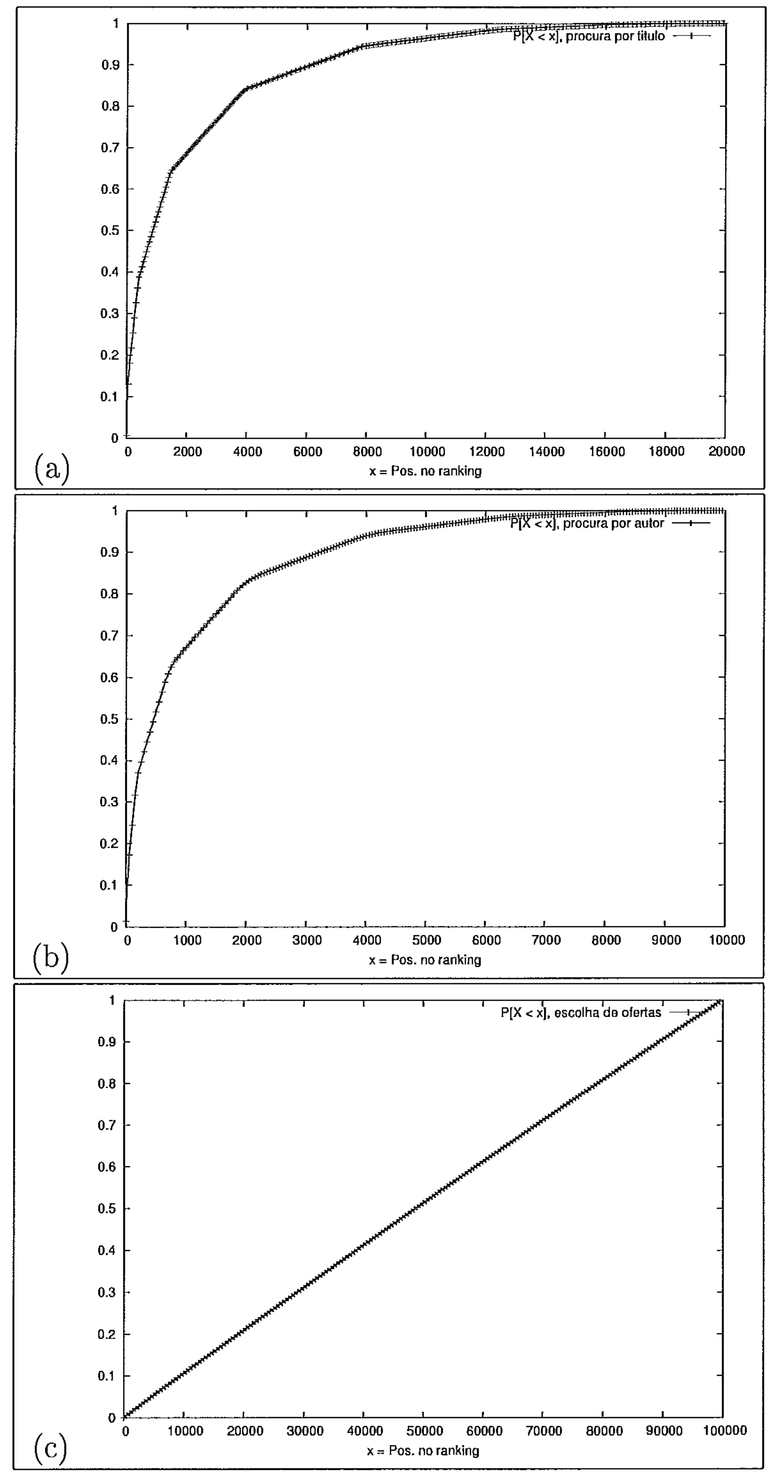

Figura 4.8: Distribuição acumulada da probabilidade de ocorrência das strings de procura por título (a) e autor (b), e da ocorrência das ofertas (c), conforme a posição no ranking, em 100.000.000 de requisições, para um banco de dados de 100k itens.

de um período de aquecimento de 30 minutos. A duração de cada experimento continuou sendo de 30 minutos.

A carga foi diminuída para 50 navegadores para manter a ocupação de CPU do SGBD dentro de uma faixa que não causasse saturação, o que impediria a comparação de ocupação de CPU entre as três cargas.

4.5.1 Desempenho

A figura 4.9 mostra os resultados das medidas de interações web por minuto. Há ganhos de desempenho para todas as cargas nesse cenário, sendo o ganho mais expressivo o de 129% para a carga "Browsing Mix", e os menos importantes de 8% para a carga "Shopping Mix" e 7% para a carga "OLTP Mix".

Vazão e ocupação de CPU

Novamente há correspondência entre o ganho de desempenho no uso de cache e a taxa de ocupação de CPU do servidor de banco de dados, que pode ser verificado comparando-se os resultados de desempenho com a figura 4.10(a). O custo adicional de operação da cache pelo *driver* ainda pode ser verificado na figura 4.10(b), que mostra a ocupação da CPU do servidor de aplicação, mas tem impacto menor na comparação com a ocupação da primeira série de experimentos, registrada na figura 4.3(b).

A menor ocupação de CPU para o servidor de aplicação ocorre porque a carga gerada na segunda série de experimentos é de apenas 50 navegadores, contra 300 navegadores da carga gerada na primeira série, e por isso a carga sobre o servidor de aplicação é menor. Como também é menor a carga sobre o servidor web, o que pode ser confirmado comparando-se as figuras 4.3(c), que mostra a ocupação de CPU do servidor web na primeira série de experimentos, e 4.10(c), que mostra a ocupação na segunda série.

Fator de escala, peso das consultas e vazão

A queda na ocupação de CPU no servidor de banco de dados, entretanto, não foi acompanhada de maneira tão acentuada, como ocorreu para os outros servidores. Isso é constatado na comparação das duas séries de experimentos, entre as figuras 4.3(a) e 4.10(a). O motivo é que a carga sobre o banco de dados no fator de escala de 1 milhão de itens é maior do que a no fator de escala de 100 mil itens para algumas consultas.

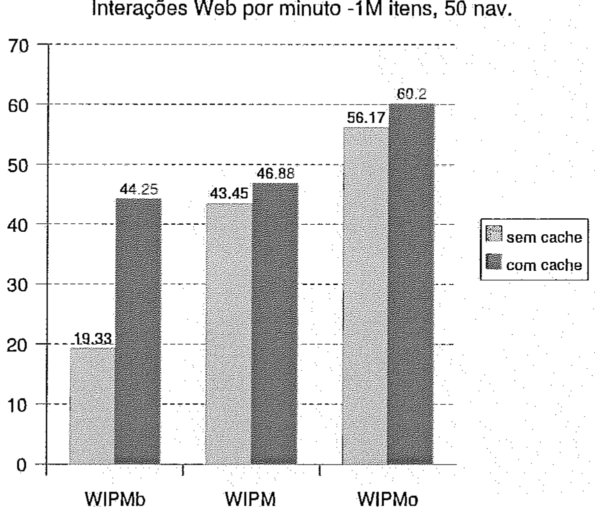

Figura 4.9: Interações Web por minuto, com e sem cache, usando-se as cargas Browsing (WIPMb), Shopping (WIPM) e OLTP (WIPMo) - segunda série

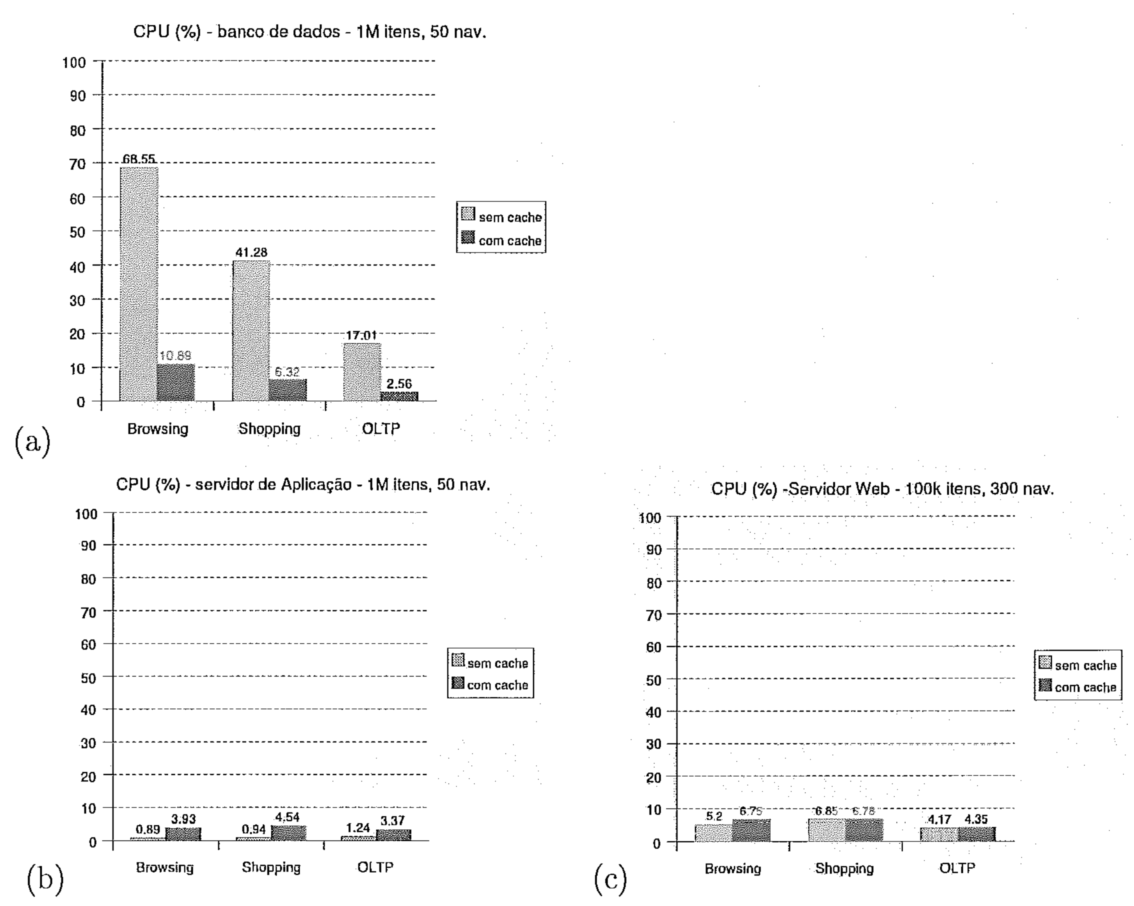

Figura 4.10: Taxa de ocupação da CPU das máquinas com o servidor de banco de dados (a), servidor de aplicação (b) e servidor Web (c) - segunda série

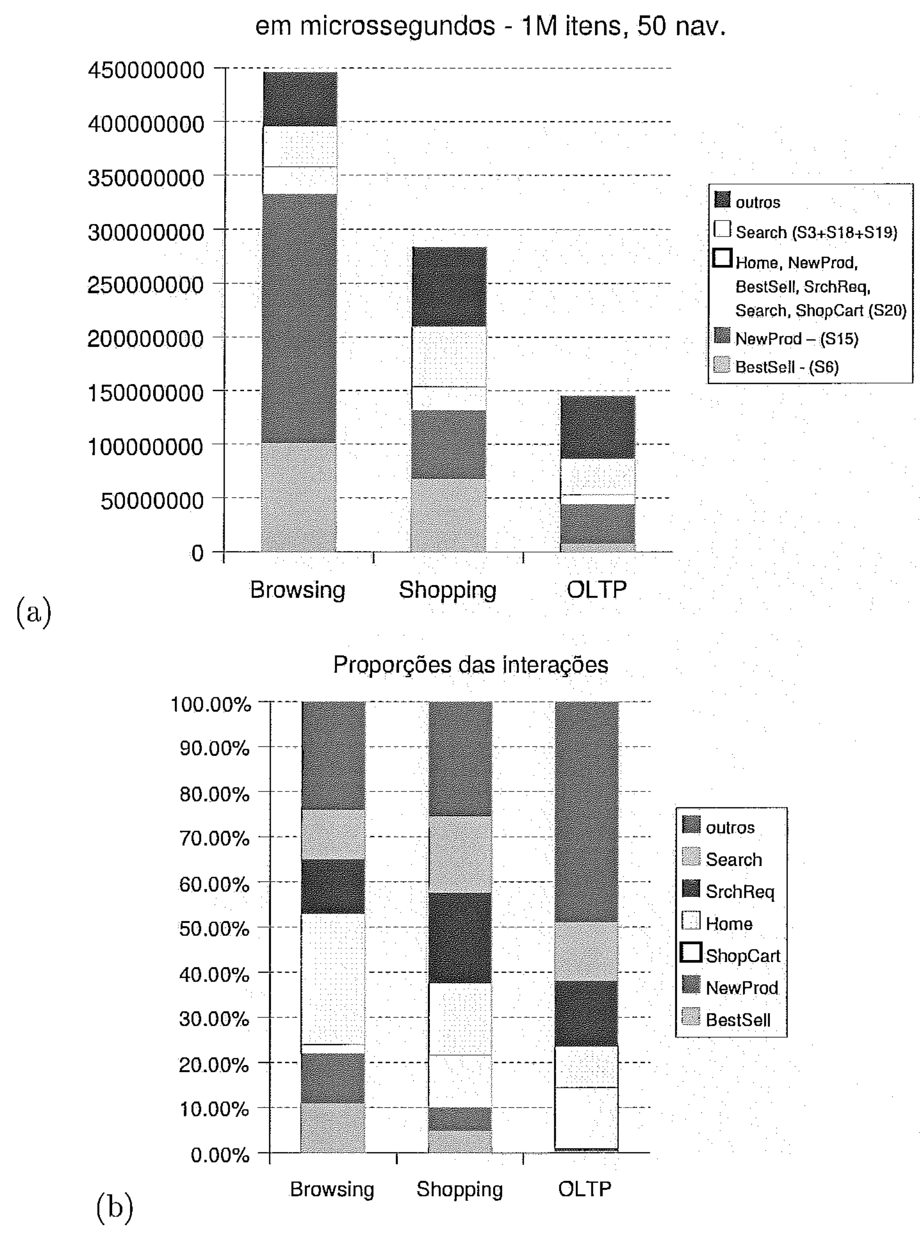

Figura 4.11: Custo de falhas fracionado entre interações, em microssegundos (a) e fracionamento da ocorrência das interações (b) - segunda série

	Browsing	Shopping	OLTP
primeira série (100 mil itens)	73%	61%	52%
segunda série (1 milhão de itens)	94%	93%	65%

Tabela 4.6: Redução do tempo aguardando banco de dados devido ao uso da cache

A figura 4.11(a), que mostra o custo de falhas fracionado entre as consultas, ajuda a explicar esse fato. Na segunda série as consultas de tipo S6 e S15 continuam entre as que mais contribuem para o custo total de falhas, mas as consultas S18 e S19, que envolvem procuras por palavras chave, ganham importância no custo total de falhas. Isso se deve ao fato de que as consultas que procuram entre 1 milhão de itens são mais demoradas que as que procuram entre 100 mil itens, por melhor que seja o sistema de indexação e o algoritmo de busca, se não pelo custo de processamento, em função do maior volume e dispersão de localização dos dados em disco.

As consultas sujeitas a um custo maior de falhas devido ao fator de escala de 1 milhão de itens têm em contrapartida impacto positivo quando ocorrem acertos em cache para tais consultas. Por isso a diferença de tempo de espera pelo SGBD entre os experimentos sem uso de cache e com uso de cache é maior nesse fator de escala. A tabela 4.6 mostra a redução do tempo de espera por banco de dados devido ao uso da cache para a primeira série, com 100 mil itens, e para a segunda, com 1 milhão de itens.

A redução do tempo de espera pelo banco de dados é maior na segunda série em todas as três cargas, apesar de nessa série a figura 4.10(a) mostrar que a ocupação de CPU para o banco de dados é menor em comparação com figura 4.3(a) da primeira série. Isso sugere que há consultas mais demoradas na série de 1 milhão de itens que se beneficiam mais do uso da cache quando há acertos do que na série de 100 mil itens.

4.6 Discussão

Os experimentos indicam que a eficiência de uma cache de banco de dados transparente e com garantia de consistência, para uso em servidores de comércio eletrônico e implementada a partir do modelo do dCache, depende da relação entre operações de leitura e de escrita no banco de dados, mas também depende principalmente da ocupação do servidor de banco de dados.

Os experimentos foram realizados em um cluster - um ambiente de computação

paralela que dispõe de rede gigabit de alta velocidade - e o SGBD utilizado, o MySQL usando o motor de armazenamento MyISAM, apesar de ser desprovido de algumas características desejáveis presentes em outros servidores de bancos de dados, é considerado rápido.

Além disso, foram feitas adaptações na implementação disponível do benchmark TPC-W de modo a tornar as consultas mais rápidas para o SGBD MySQL. Por exemplo o uso, em consultas SQL de busca por palavras-chave, do operador "match...against" associado a um índice de busca textual, no intuito de tornar a comparação com o uso de cache mais justa.

Em um cenário tecnologicamente realista, mas pouco favorável ao uso de cache nos moldes do dCache, a vantagem evidente de uso da cache ocorre quando o SGBD é um gargalo, e quando as consultas se repetem, beneficiando-se da cache. Para o benchmark TPC-W, o SGBD pode ser um gargalo quando as consultas são complexas, envolvendo junções, agregações e ordenações, o que ficou demonstrado da primeira série de experimentos com a carga gerada por um grande número de navegadores. O SGBD também pode ser um gargalo quando as consultas envolvem buscas em tabelas com muitas linhas, o que foi demonstrado na segunda série de experimentos, com um fator de escala grande para o número de itens.

Também ficou demonstrado que as diferentes consultas causam ocupações diferentes em memória na cache, devido a fatores como a freqüência de uso da consulta, o número de invalidações que ela sofre, e a distribuição estatística dos parâmetros que ocorrem para cada tipo de consulta.

Capítulo 5

Trabalhos relacionados

A preocupação em prover mecanismos que poupem o programador de aplicações para a Web de lidar com os detalhes de uso de implementações de cache para banco de dados aparece em trabalhos acadêmicos e produtos comerciais.

Degenaro et al.[21] demonstrou a aplicação do *General-Purpose Software Cache* (GPS) em uma versão melhorada de um mecanismo chamado *data update propagation* (DUP) para manter uma cache atualizada em relação ao banco de dados. Um grafo dirigido de dependência, chamado de *object dependence graph* (ODG) é mantido entre dados em cache para determinar quais dados são afetados por mudanças no banco de dados. Os valores antigos e os novos valores de atributos modificados são considerados para se determinar como são feitas as atualizações de dados em cache, por meio de anotações feitas nas arestas do grafo de dependência representando valores que ocorrem nas consultas feitas.

Entretanto, diferente do dCache, que compila as consultas em tempo de execução, os ODG são gerados a partir das consultas em tempo de compilação, o que em um ambiente real implicaria que o programador da aplicação submetesse seu código fonte a uma etapa adicional de processamento. O programador também é assim poupado de operar a cache em seu código fonte.

Luo et al. [14] usam a funcionalidade de processamento distribuído[1] do SGBD IBM DB2 para implementar uma cache chamada DBCache, que é uma instância do servidor de banco de dados sem acesso a disco, mas apenas com acesso a memória primária. O DBCache é uma cache de tabelas associada a um mecanismo de processamento distribuído de consultas. Com base numa estimativa de custo de execução, o DBCache decide se a execução da consulta ocorre na própria instância local do DBCache, com acesso a cache, ou em uma outra instância tradicional do DB2. Além disso, pode haver particionamento de execução de uma mesma consulta

[1] Em inglês, *federated database functionality*.

entre duas instâncias. Também não são necessárias modificações na aplicação, que enxerga o DBCache como mais uma instância do servidor DB2.

Cecchet et al. [13] implementam o C-JDBC, um *driver* JDBC, como uma alternativa a bancos de dados paralelos. O C-JDBC é um *middleware* posto entre a aplicação e vários servidores de banco de dados. Ele realiza o balanceamento e distribuição de carga de forma transparente, sem necessidade de alteração da aplicação, e implementa tolerância a falhas entre várias instâncias de SGBD heterogêneos, bastando para isso que haja *drivers* JDBC disponíveis para esses SGDB. Esse *driver* também permite que outras instâncias do *driver* C-JDBC sejam tratadas como SGBD, permitindo assim arranjos hierárquicos. Esses arranjos são chamados de RAIDb (*Redundant Array of Inexpensive Databases*).

Apesar de a função primordial do C-JDBC ser a de balanceador de carga, ele também inclui a funcionalidade opcional de uma *cache* de consultas, como o dCache. Diversas opções de granularidade de consistência de *cache* são implementadas, como invalidação total dos dados, invalidação baseada em tabelas e invalidações baseadas em colunas. Os resultados de desempenho no uso da função de balanceamento de carga do C-JDBC apresentados para o TPC-W [13], no fator de escala de 10.000 itens, mostram em média ganhos em aceleração (*speedup*) quase linear para a medida de vazão "pedidos por minuto", na proporção em que se adicionam mais nós servidores de bancos de dados, até o limite de 6 nós. Não são mostrados entretanto detalhes sobre tal medida de vazão, como por exemplo se ela corresponde à medida oficial WIPS do TPC-W, e infelizmente o uso opcional de cache não é avaliado com esse benchmark.

O uso de *cache* no C-JDBC é avaliado [13] apenas com o benchmark RuBiS (Rice University Bidding System) [49], que modela um serviço de leilão eletrônico. São medidos a vazão, também em "pedidos por minuto", que tem uma melhora de apenas 7,5% com o uso de *cache*, o tempo de resposta do banco de dados, que é reduzido com o uso de *cache* por um fator de quase 3, e a ocupação de CPU da máquina com o SGBD, que é reduzida de 100% para 85% com o uso de *cache*. Esses resultados são para o caso de *cache* coerente (não relaxada) como é o caso do dCache.

Um trabalho importante, em parte motivador da origem deste trabalho, é o da análise do eCache [9], que é uma implementação de loja virtual para um cluster que explora uma cache cooperativa. O eCache usa o modelo de programação por memória compartilhada distribuída para manter um diretório de metadados que informa a localização de um dado em *cache* no espaço agregado das *caches* dos vários servidores paralelos. Apesar de não considerar os problemas de consistência

de *cache* resultantes das operações de escrita, o eCache mostra que é possível obter ganhos de desempenho no servidor de comércio eletrônico implementado usando-se uma *cache* cooperativa, reduzindo substancialmente nesse cenário a carga imposta ao servidor de banco de dados.

Do ponto de vista da organização da *cache*, os modelos de *cache* semântica [50, 51] e de cacheamento de espaço multidimensional em pedaços (*chunks*)[52] são alternativas a *cache* de tabelas e a *cache* de consultas e tuplas, como é o caso do dCache. No modelo semântico, uma consulta pode ser parcialmente atendida por dados localizados em *cache*, e parcialmente pelos dados buscados no SGBD por uma consulta especialmente criada para buscar os dados residuais, ou seja, o conjunto complementar de dados que não foram encontrados em *cache*. Os dados armazenados em *cache* são agrupados em regiões semânticas, sendo que cada região semântica é associada a uma fórmula, que descreve como os dados se restringem com respeito a faixas de valores dos atributos.

O modelo de cacheamento multidimensional em pedaços é semelhante ao de *cache* semântica. Entretanto, em vez de agrupar os dados em *cache* em regiões semânticas de tamanho variável e de forma dinâmica, o espaço multidimensional dos dados retornados pelas consultas é dividido em pedaços uniformes de tamanho estático. Essa uniformidade visa a tornar o mapeamento das regiões semânticas e o reuso de consultas menos complexo.

O modelo de *cache* semântica foi avaliado por Soundararajan et al.[51] com o *benchmark* TPC-W buscando o objetivo de melhorar o desempenho da *cache* de consultas transparente do C-JDBC. De interesse particular é a redução do custo de invalidação devido a comandos INSERT, quando comparado ao custo de invalidação na política do C-JDBC de invalidação baseada em coluna. As novas tuplas que surgem devido a comandos INSERT são mantidas em tabelas temporárias. Ao receber uma nova consulta, o mecanismo da cache semântica divide a consulta em dois tipos: a consulta original direcionada às tabelas reais e uma ou mais consultas residuais direcionadas às tabelas temporárias. O resultado final da nova consulta recebida é obtido por meio da união dos resultados desses dois tipos de consulta, evitando-se em alguns casos a invalidação causada por comandos INSERT.

Resultados experimentais do *benchmark* TPC-W, no fator de escala de 100 mil itens e 1000 navegadores (2,8 milhão de clientes), na comparação entre a política básica de invalidação de cache por coluna do C-JDBC e a política de invalidação com cache semântica, apontam uma redução no tempo de resposta de página, em favor da cache semântica, de um fator de 2,9 para a carga Shopping Mix e redução de um fator de 1,2 para Browsing Mix. Entretanto o aumento de vazão não passa

de 10% para ambas as cargas. A comparação com os resultados apresentados para o dCache é prejudicada porque a metodologia experimental utilizada com a *cache* semântica não incluiu um caso base sem *cache*, além de a configuração dos experimentos da *cache* semântica ter sido feita com o emulador de navegadores, servidor Web e de aplicação em um único nó, a *cache* como *middleware* em um segundo nó, distante do servidor de aplicação, e o SGBD em um terceiro nó.

Capítulo 6

Conclusões

Na presente dissertação analisaram-se os resultados da aplicação, em um servidor de comércio eletrônico, de um *driver* projetado e implementado para armazenar em cache os resultados de consultas a um banco de dados, com manutenção automática e transparente de consistência.

Para atingir esse objetivo, foi incluído no *driver* um mecanismo de *parsing* e modificação das consultas e comandos de atualização de dados em SQL, de forma a garantir que os dados presentes em cache tenham índices, ou chaves, que possibilitem atualização individual de dados em cache em caso de atualizações no banco de dados. Foram incluídos também um mecanismo para invalidação de dados por tipo de consulta, além de um mecanismo de suspensão temporária de uso da cache com o objetivo de manter um grau de isolamento entre transações. Os três mecanismos contribuem juntos para garantir a consistência dos dados em cache em relação ao banco de dados.

Os experimentos realizados em um ambiente de computação paralela com o benchmark TPC-W, que modela uma loja virtual de livros para a Web, mostraram que a relação entre operações de leitura e escrita é importante no desempenho do *driver*, mas que taxa de ocupação do servidor de banco de dados é bem mais significativa nesse contexto. Quando essa taxa é alta, em função da ocorrência numerosa de consultas complexas ou que envolvam muitos registros, o desempenho da cache tende a ser melhor.

Constatou-se pelos experimentos que o acesso a banco de dados é apenas um dos componentes que têm efeito na vazão total do servidor. A redução do tempo de acesso ao banco de dados não se traduziu em aumento expressivo da vazão total do servidor em todos os casos, principalmente quando houve aumento da ocupação de CPU da máquina com o servidor de aplicação, em função do custo adicional de processamento da cache no *driver*, que também funciona nesse servidor, e da

competição desse processamento com o próprio servidor de aplicação.

Quanto à taxa de operações de leitura e escrita, verificou-se que mesmo uma taxa pequena de escritas pode causar um grande número de falhas em cache em função do modelo de invalidações adotado no *driver*, que elimina de uma vez todas as entradas em cache que afetam cada tipo de consulta. Um modelo de invalidações com mais granularidade, a exemplo do que foi usado no mesmo *driver* para as atualizações - individuais para cada tupla - pode ser uma solução futura para esse tipo de problema, e de implementação relativamente fácil, uma vez que o mecanismo de *parsing*, modificação e inclusão de chaves para todos os comandos SQL já se encontra implementado no *driver*.

Está em vista como investigação futura a avaliação do uso do *driver* com outros SGBD que sejam rápidos como o MySQL com tabelas MyISAM, mas que não apresentem as limitações deste servidor de bancos de dados, como a ausência de suporte a consistência de transações. O MySQL aceita a sintaxe usada para transações utilizada no benchmark, mas ignora silenciosamente o isolamento transacional. O benchmark mantém consistência entre transações do banco de dados por meio de *locks*, e o *driver* mantém o isolamento em cache por meio da suspensão temporária do uso da cache. Os efeitos do isolamento transacional centrado no SGBD precisam ser investigados. Além disso, uma melhor investigação sobre a sintonia fina do MySQL, para aumentar o desempenho das consultas para a aplicação TPC-W em particular precisa ser realizada.

A constatação de que o desempenho do *driver* é melhor quando a taxa de ocupação do servidor de banco de dados é alta sugere como direção futura de investigação a inclusão no *driver* de um mecanismo de propagação das atualizações e invalidações para outras instâncias do *driver* em servidores de aplicação paralelos em um agregado de máquinas, como ilustra a figura 6.1. Com isso, seria aliviada a maior tendência de gargalo do SGBD. A escalabilidade do desempenho em um ambiente seguindo esse modelo, em função das mensagens trocadas entre nós do agregado, seria uma das questões principais a serem estudadas.

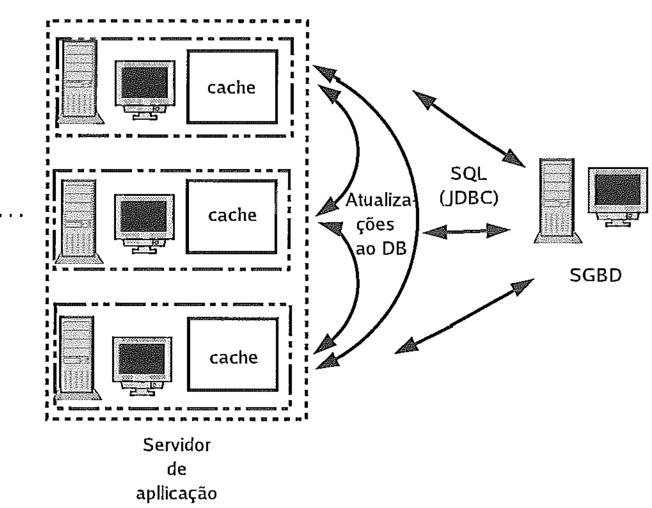

Figura 6.1: Arquitetura de cluster com o dCache e vários servidores de aplicação

Apêndice A

Gramática do analisador sintático

Este apêndice contém a descrição da sintaxe do SQL reconhecido pelo dCache. A gramática foi construída a partir de um subconjunto do SQL 92 usado pelo MySQL, usando-se as definições do manual de referência deste SGBD [44].

O formato da descrição é o da entrada do gerador de parsers Javacc [41]. O parser gerado para esta gramática pelo Javacc é do tipo *top-down* que lê gramáticas LL(1), ou seja, determinístico descendente, com leitura da entrada da esquerda para a direita, e produção de uma derivação mais à esquerda [42, 43] . Em algumas produções, entretanto, existem ambigüidades que são resolvidas transformando-se a gramática em LL(k) nesses pontos, o que na implementação foi feito nesses lugares com um marcador "lookahead (k)".

A notação das expressões regulares é a tradicional:

- "A | B" significa "A ou B"
- "A?" significa "A é opcional"
- "A*" significa "0 ou mais ocorrências de A"
- "A+" significa "1 ou mais ocorrências de A"

Para facilitar a legibilidade, não são incluídas nesta listagem nem as ações semânticas nem os marcadores "lookahead".

A.1 Regras de produção

```
StatementClass Input() : {
      SelectStatement()
      | UpdateStatement()
      | InsertStatement()
```

```
    | DeleteStatement()
    | OtherStatements()
  }

  SelectStatement() : {
    <SELECT> ( <ALL> | <DISTINCT> )?
    ( <FROM> TableList() ( <WHERE> SearchCondition() )? )?
    ( <GROUP> <BY> GroupByList() )?
    ( <ORDER> <BY> OrderByList() )?
    ( <LIMIT> LimitRange() )?
  }

  UpdateStatement() : {
    <UPDATE> TableList() <SET> SetList() ( <WHERE> SearchCondition()
)?
  }

  InsertStatement() : {
    <INSERT> (<INTO>)? TableName() ( "(" ColumnList() ")" )?
    ( ( <VALUES> "(" InsertValuesList() ")" )
     | ( <SET> SetList() )
     | ( SelectStatement() )
    )
  }

  DeleteStatement() : {
    <DELETE> <FROM> TableName() ( <WHERE> SearchCondition() )?
  }

  OtherStatements() : {
    <CREATE> { return; } // Apenas para
    | <DROP> { return; } // classificar
  }

  InsertValuesList() : {
    ( Expression() | <DEFAULT_TOKEN> )
    ( ( "," (Expression() | <DEFAULT_TOKEN> )) )*
  }

  LimitRange() : {
    <INT_LITERAL> "," <INT_LITERAL>
    | <INT_LITERAL> <OFFSET> <INT_LITERAL>
    | <INT_LITERAL>
  }

  SelectExpressionList() : {
```

```
  SelectExpression() ( (<AS>)? SelectAlias() )?
  ( "," SelectExpression() ( (<AS>)? SelectAlias() )? ) )*
}

SelectExpression() : {
  Expression() | "*"
}

SelectAlias() : {
  <USER_DEF_NAME>
}

Function() : {
  <USER_DEF_NAME> "(" ParameterList() ")"
}

ParameterList() : {
  "*" | ( "," Parameter() )*
}

Parameter() : {
  <INTERVAL> Expression() <USER_DEF_NAME>
  | Expression()
}

SearchCondition() : {
  BooleanExpression()
}

BooleanExpression() : {
  SimpleBooleanExpression()
  ( BooleanOperator() SimpleBooleanExpression() )*
}

SimpleBooleanExpression() : {
  "(" BooleanExpression() ")"
  | MatchAgainstExpression()
  | SimpleExpression() RelationalOperator() SimpleExpression()
}

MatchAgainstExpression() : {
  <MATCH> "(" ColumnList() ")"
  <AGAINST> "(" ( QuotedString() | DynamicParameter() )
  ( <IN> <BOOLEAN> <MODE> )? ")"
}
```

```
OrderByList() : {
  ColumnName() (<ASC> | <DESC> )?
  ( "," ColumnName() ( <ASC> | <DESC> )? )*
}

GroupByList() : {
  ColumnName() ( "," ColumnName() )*
}

SetList() : {
  ColumnName() "=" Expression()
  ( "," ColumnName() "=" Expression() )*
}

TableList() : {
  TableIdentifier() ( "," TableIdentifier() )*
}

TableIdentifier() : {
  TableName() <AS> TableAlias()
  | TableName() TableAlias()
  | TableName()
}

TableAlias() : {
  <USER_DEF_NAME>
}

TableName() : {
  <USER_DEF_NAME>
}

ColumnName() : {
  TableName() "." "*"
  | ( TableName() "." )? ColumnIdentifier()
}

ColumnList() : {
  ColumnName() ( "," ColumnName() )*
}

ColumnIdentifier() : {
  <USER_DEF_NAME>
}
```

```
RelationalOperator() : {
  "<" ">"
  | "<" "="
  | "<"
  | "="
  | ">" "="
  | ">"
  | "!" "="
  | <LIKE>
  | <NOT> <LIKE>
}

BooleanOperator() : {
  <AND> | <OR>
}

AddingOperator() : {
  "+" | "-"
}

MultiplyingOperator() : {
  "*" | "/"
}

Expression() : {
  SimpleExpression() ( RelationalOperator() SimpleExpression() )?
}

SimpleExpression() : {
  ( AddingOperator() )? Term() ( AddingOperator() Term() )*
}

Term() : {
  Factor() ( MultiplyingOperator() Factor() )*
}

Factor() : {
  Number()
  | Function()
  | ColumnName()
  | "(" Expression() ")"
  | QuotedString()
  | DoubleQuotedString()
  | DynamicParameter()
}
```

```
DynamicParameter() : {
  <DYNAMIC_PARAMETER>
}

QuotedString() : {
  <QUOTED_CHAR_STRING_LITERAL>
}

DoubleQuotedString() : {
  <DOUBLE_QUOTED_CHAR_STRING_LITERAL>
}

Number() : {
  <FP_LITERAL>
  | <INT_LITERAL>
}
```

A.2 Tokens

Obs.: Os tokens foram configurados para serem reconhecidos ignorando distinção entre maiúsculas e minúsculas.

```
SKIP : { " " | "\t" | "\n" | "\r" }

TOKEN : {
  < SELECT: "select" >
  | < ALL: "all" >
  | < DISTINCT: "distinct" >
  | < FROM: "from" >
  | < WHERE: "where" >
  | < AND: "and" >
  | < OR: "or" >
  | < LIKE: "like" >
  | < MATCH: "match" >
  | < AGAINST: "against" >
  | < IN: "in" >
  | < BOOLEAN: "boolean" >
  | < MODE: "mode" >
  | < NOT: "not" >
  | < ORDER: "order" >
  | < GROUP: "group" >
  | < BY: "by" >
  | < ASC: "asc">
  | < DESC: "desc">
```

```
| < LIMIT: "limit" >
| < OFFSET: "offset" >
| < UPDATE: "update" >
| < SET: "set" >
| < AS: "as">
| < INSERT: "insert" >
| < INTO: "into" >
| < VALUES: "values" >
| < INTERVAL:"interval" >
| < DELETE: "delete" >
| < CREATE: "create" >
| < DROP: "drop" >
| < DEFAULT_TOKEN: "default" >
| < DYNAMIC_PARAMETER: "?" >
|< USER_DEF_NAME: ["a"-"z","A"-"Z","_","$"] ( ["a"-"z","A"-"Z","_","0"-
"9"] )* >
| < QUOTED_CHAR_STRING_LITERAL: "'" (~["'"])* "'" >
| < DOUBLE_QUOTED_CHAR_STRING_LITERAL: "\"" (~["\""])*
"\"" >
| < FP_LITERAL: (["0"-"9"])+ "." (["0"-"9"])* >
| < INT_LITERAL: ( ["0"-"9"] )+ >
}
```

Apêndice B

Interface gráfica

Apesar de não ser comum a um *driver* possuir uma interface gráfica, o dCache pode opcionalmente apresentar uma interface para propósitos de depuração. Para isso basta que a aplicação informe a opção "showGUI" na URL de conexão ao banco de dados, como no exemplo abaixo:

jdbc:dcache:///develop;jdbc:mysql://localhost/tpcw;showGUI

Após a primeira conexão com o banco de dados, a interface gráfica mostra uma organização hierárquica que enumera instâncias diferentes das caches e demais estruturas internas do dCache. É possível haver mais de uma instância para que várias aplicações possam se valer de caches isoladas, em um mesmo servidor de aplicação. O que difere uma instância da outra é a URL de conexão ao banco.

Para cada uma das instâncias, três tipos de informação estão disponíveis:

1. O conteúdo da "Statement Cache": pode-se examinar a árvore sintática de cada objeto "StatementClass", que corresponde aos comandos SQL em cache, o que pode ser visto no detalhe da figura B.2.

2. O conteúdo da "Param Cache", ou cache de parâmetros, que contém as relações resultantes do uso conjugado dos comandos SQL e dos parâmetros, como mostra o detalhe da figura B.2.

3. O schema do banco de dados, o que inclui as chaves primárias de cada tabela, conforme o detalhe da figura B.1.

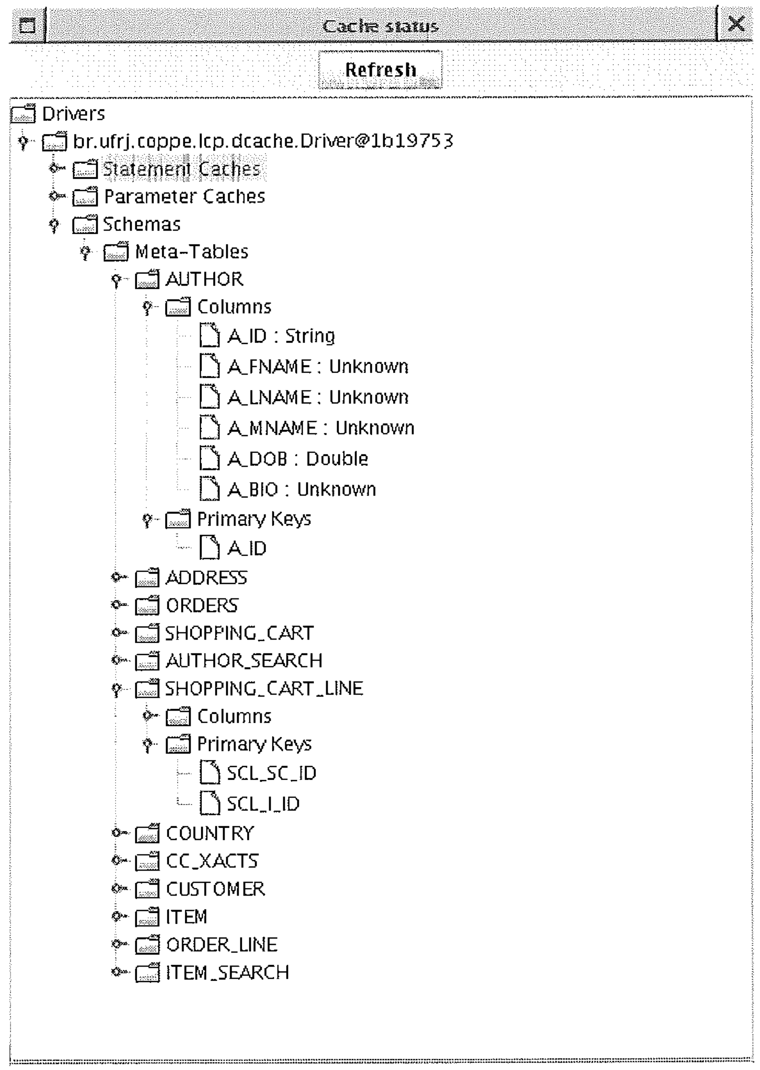

Figura B.1: Schema do banco de dados em detalhe

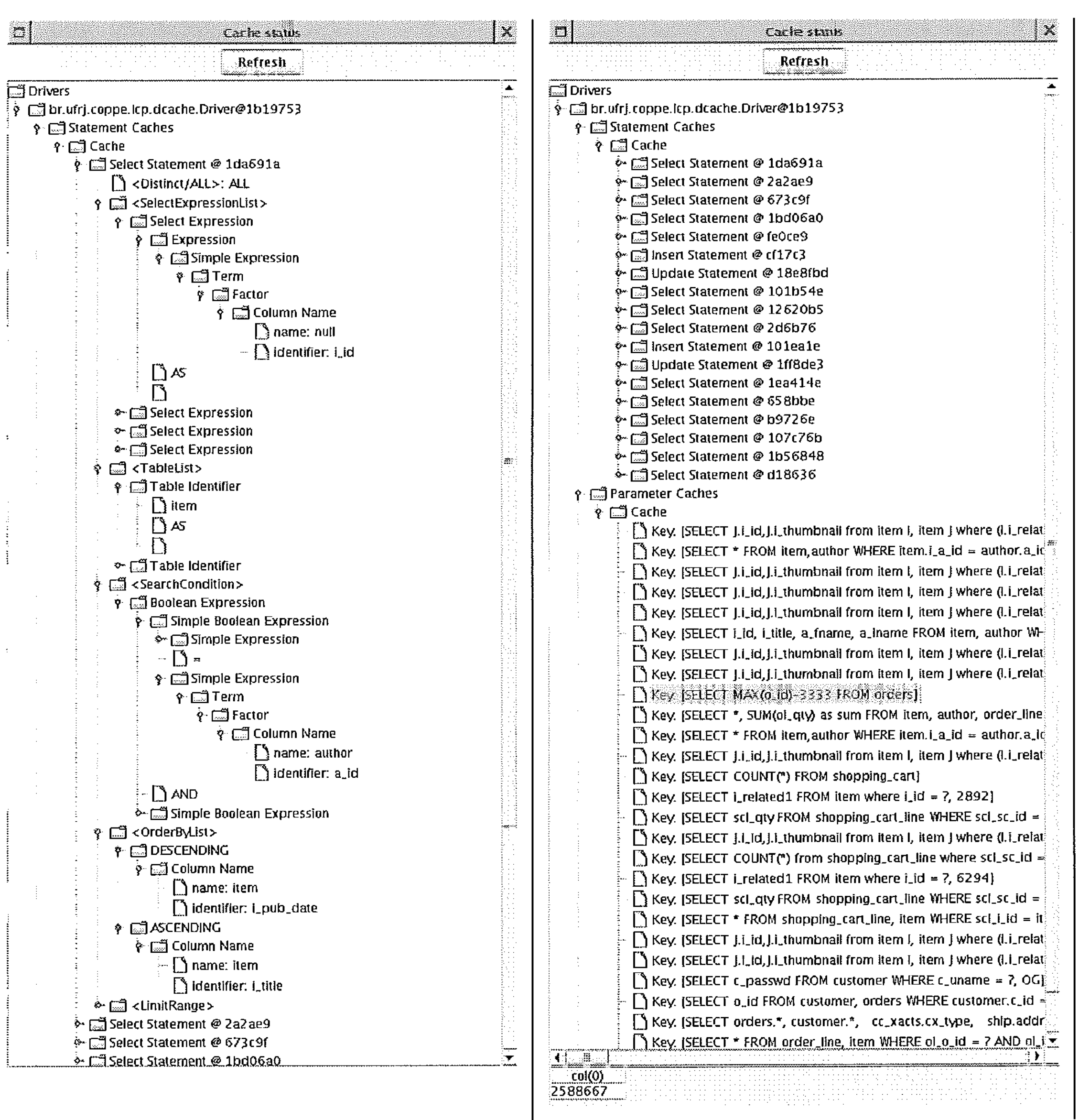

Figura B.2: À esquerda, a árvore sintática de uma consulta SQL em detalhe, à direita a cache de parâmetros em detalhe.

Apêndice C

Dados experimentais em detalhe

Nome da tabela	série 1 (registros)	série 1 (MB)	série 2 (registros)	série 2 (MB)
address	1.728.000	232	288.000	39
author	25.000	9	250.000	97
author_search	25.000	1	250.000	10
cc_xacts	777.600	83	129.600	15
country	92	0,008	92	0,008
customer	864.000	382	144.000	63
item	100.000	62	1.000.000	624
item_search	100.000	11	1.000.000	110
order_line	2.332.890	313	387.573	53
orders	777.600	81	129.600	15
Total (MB)		1.174		1.026

Tabela C.1: Tamanho das tabelas utilizadas na série 1 (100k itens, 300 nav.) e série 2 (1M itens, 50 nav.) de experimentos

Abreviatura	Nome da interação no tpc-w
Home	Home web interaction
NewProd	New Products web interaction
BestSell	Best Sellers web interaction
Detail	Product Detail web interaction
Search	Search Result web interaction
SrchReq	Search Request web interaction
OrdDisp	Order Display web interaction
ShopCart	Shopping Cart web interaction
BuyReq	Buy Request web interaction
BuyConf	Buy Confirm web interaction
AdmReq	Admin Request web interaction
AdmConf	Admin Confirm web interaction

Tabela C.2: Abreviaturas usadas para as interações Web

S1: ocorre em: BuyReq

```
SELECT *
FROM customer, address, country
WHERE customer.c_addr_id = address.addr_id
   AND address.addr_co_id = country.co_id
   AND customer.c_uname = ?
```

S2: ocorre em: ShopCart

```
SELECT *
FROM item
WHERE i_id = ?
```

S3: ocorre em: Search

```
SELECT *
FROM item, author
WHERE item.i_a_id = author.a_id
   AND item.i_subject = ?
ORDER BY item.i_title
LIMIT 0,50
```

S4: ocorre em: Detail, AdmReq, AdmConf

```
SELECT *
FROM item,author
WHERE item.i_a_id = author.a_id
   AND i_id = ?
```

S5: ocorre em: OrdDisp

```
SELECT *
FROM order_line, item
WHERE ol_o_id = ?
   AND ol_i_id = i_id
```

S6: ocorre em: BestSell

```
SELECT *, SUM(ol_qty) as sum
FROM item, author, order_line
WHERE item.i_id = order_line.ol_i_id
   AND item.i_a_id = author.a_id
   AND order_line.ol_o_id > ?
   AND item.i_subject = ?
GROUP BY i_id, i_title, a_fname, a_lname
ORDER BY sum DESC
LIMIT 50
```

S7: ocorre em: BuyConf

```
SELECT addr_co_id
FROM address
WHERE addr_id = ?
```

S8: ocorre em: BuyReq, BuyConf

```
SELECT addr_id
FROM address
WHERE addr_street1 = ?
   AND addr_street2 = ?
   AND addr_city = ?
   AND addr_state = ?
   AND addr_zip = ?
   AND addr_co_id = ?
```

S9: Ocorre em: BuyConf

```
SELECT c_addr_id
FROM customer
WHERE customer.c_id = ?
```

S10: Ocorre em: BuyConf

```
SELECT c_discount
FROM customer
WHERE customer.c_id = ?
```

S11: Ocorre em: Home

```
SELECT c_fname,c_lname
FROM customer
WHERE c_id = ?
```

S12: Ocorre em: OrdDisp

```
SELECT c_passwd
FROM customer
WHERE c_uname = ?
```

S13: Ocorre em: BuyReq, BuyConf

```
SELECT co_id
FROM country
WHERE co_name = ?
```

Figura C.1: Comandos S1 a S13

S14: Ocorre em: BuyConf

```
SELECT count(o_id)
FROM orders
```

S15: Ocorre em: NewProd

```
SELECT i_id, i_title, a_fname, a_lname
FROM item, author
WHERE item.i_a_id = author.a_id
  AND item.i_subject = ?
ORDER BY item.i_pub_date DESC,item.i_title
LIMIT 0,50
```

S16: Ocorre em: ShopCart

```
SELECT i_related1
FROM item where i_id = ?
```

S17: Ocorre em: BuyConf

```
SELECT i_stock
FROM item
WHERE i_id = ?
```

S18: Ocorre em: Search

```
SELECT i.*, a.*
FROM author a, item i, author_search a_search
WHERE match(a_search.a_lname) against (?)
   AND i.i_a_id = a.a_id
   AND a.a_id=a_search.a_id
ORDER BY i.i_title
LIMIT 0,50
```

S19: Ocorre em: Search

```
SELECT i.*, a.*
FROM item i, item_search i_search, author a
WHERE i.i_a_id = a.a_id
   AND i_search.i_id=i.i_id
   AND match (i_search.i_title) against (? in boolean mode)
ORDER BY i.i_title
LIMIT 0,50
```

S20: Ocorre em: BestSell, Search,
Home, NewProd,
SrchReq, ShopCart

```
SELECT J.i_id,J.i_thumbnail
FROM item I, item J
WHERE (I.i_related1 = J.i_id
     OR I.i_related2 = J.i_id
     OR I.i_related3 = J.i_id
     OR I.i_related4 = J.i_id
     OR I.i_related5 = J.i_id)
     AND I.i_id = ?
```

S21: Ocorre em: BuyConf, BuyReq

```
SELECT max(addr_id)
FROM address
```

S22: Ocorre em: BuyReq

```
SELECT max(c_id)
FROM customer
```

S23: Ocorre em: BestSell

```
SELECT MAX(o_id)-3333
FROM orders
```

S24: Ocorre em: OrdDisp

```
SELECT o_id
FROM customer, orders
WHERE customer.c_id = orders.o_c_id
   AND c_uname = ?
ORDER BY o_date, orders.o_id DESC
LIMIT 0,1
```

Figura C.2: Comandos S14 a S24

S25: Ocorre em: AdmConf

```
CREATE TEMPORARY TABLE $tmp_adminXX TYPE=HEAP
  SELECT o_id
  FROM orders
  ORDER BY o_date DESC
  LIMIT 10000


SELECT ol2.ol_i_id, SUM(ol2.ol_qty) AS sum_ol
FROM order_line ol, order_line ol2, $tmp_adminXX t
WHERE ol.ol_o_id = t.o_id
    AND ol.ol_i_id = ?
    AND ol2.ol_o_id = t.o_id
    AND ol2.ol_i_id <> ?
GROUP BY ol2.ol_i_id
ORDER BY sum_ol DESC
LIMIT 0,5

DROP TABLE $tmp_adminXX
```

S26: Ocorre em: OrdDisp

```
SELECT orders.*, customer.*,   cc_xacts.cx_type,
      ship.addr_street1 AS ship_addr_street1,
      ship.addr_street2 AS ship_addr_street2,
      ship.addr_state AS ship_addr_state,
      ship.addr_zip AS ship_addr_zip,
      ship_co.co_name AS ship_co_name,
      bill.addr_street1 AS bill_addr_street1,
      bill.addr_street2 AS bill_addr_street2,
      bill.addr_state AS bill_addr_state,
      bill.addr_zip AS bill_addr_zip,
      bill_co.co_name AS bill_co_name
FROM customer, orders, cc_xacts,
    address AS ship,   country AS ship_co,
    address AS bill,    country AS bill_co
WHERE orders.o_id = ?
    AND cx_o_id = orders.o_id
    AND customer.c_id = orders.o_c_id
    AND orders.o_bill_addr_id = bill.addr_id
    AND bill.addr_co_id = bill_co.co_id
    AND orders.o_ship_addr_id = ship.addr_id
    AND ship.addr_co_id = ship_co.co_id
    AND orders.o c id = customer.c id
```

Figura C.3: Comandos S25 e S26

I1: Ocorre em: BuyReq, BuyConf

```
INSERT into address
    (addr_id, addr_street1, addr_street2,
     addr_city, addr_state, addr_zip, addr_co_id)
VALUES (?, ?, ?, ?, ?, ?, ?)
```

I2: Ocorre em: BuyConf

```
INSERT into cc_xacts
    (cx_o_id, cx_type, cx_num,
     cx_name, cx_expire, cx_xact_amt,
     cx_xact_date, cx_co_id)
VALUES (?, ?, ?, ?, ?, ?, ?, ?)
```

I3: Ocorre em: BuyReq

```
INSERT into customer
    (c_id, c_uname, c_passwd,
     c_fname, c_lname, c_addr_id,
     c_phone, c_email, c_since,
     c_last_login, c_login, c_expiration,
     c_discount, c_balance, c_ytd_pmt,
     c_birthdate, c_data)
VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?,
     ?, ?, ?, ?, ?, ?, ?, ?)
```

I4: Ocorre em: BuyConf

```
INSERT into order_line
    (ol_id, ol_o_id, ol_i_id,
     ol_qty, ol_discount, ol_comments)
VALUES (?, ?, ?, ?, ?, ?)
```

I5: Ocorre em: BuyConf

```
INSERT INTO orders
    (o_id, o_c_id, o_date, o_sub_total,
     o_tax, o_total, o_ship_type,
     o_ship_date, o_bill_addr_id,
     o_ship_addr_id, o_status)
VALUES (?, ?, ?, ?, 8.25, ?,
     ?, ?, ?, ?, 'Pending')
```

U1: Ocorre em: BuyReq

```
UPDATE customer
SET c_login = ?, c_expiration = ?
WHERE c_id = ?
```

U2: Ocorre em: AdmConf

```
UPDATE item
SET i_cost = ?, i_image = ?, i_thumbnail = ?,
   i_pub_date = ?, i_related1 = ?,
   i_related2 = ?, i_related3 = ?,
   i_related4 = ?, i_related5 = ?
WHERE i_id = ?
```

U3: Ocorre em: BuyConf

```
UPDATE item
SET i_stock = ?
WHERE i_id = ?
```

Figura C.4: Comandos I1 a I5 e U1 a U3

	Comando	Ocorrência
S1	SELECT * FROM customer, address, country WHERE customer.c_addr_id = address.addr_id AND address.addr_co_id ...	BuyReq
S2	SELECT * from item where i_id = ?	ShopCart
S3	SELECT * FROM item, author WHERE item.i_a_id = author.a_id AND item.i_subject = ? ORDER BY item.i_title...	Search
S4	SELECT * FROM item,author WHERE item.i_a_id = author.a_id AND i_id = ?	Detail, AdmReq, AdmConf
S5	SELECT * FROM order_line, item WHERE ol_o_id = ? AND ol_i_id = i_id	OrdDisp
S6	SELECT *, SUM(ol_qty) as sum FROM item, author, order_line WHERE item.i_id = order_line.ol_i_id AND ...	BestSell
S7	SELECT addr_co_id FROM address WHERE addr_id = ?	BuyConf
S8	SELECT addr_id FROM address WHERE addr_street1 = ? AND addr_street2 = ? AND addr_city = ? ...	BuyReq, BuyConf
S9	SELECT c_addr_id FROM customer WHERE customer.c_id = ?	BuyConf
S10	SELECT c_discount FROM customer WHERE customer.c_id = ?	BuyConf
S11	SELECT c_fname,c_lname FROM customer WHERE c_id = ?	Home
S12	SELECT c_passwd FROM customer WHERE c_uname = ?	OrdDisp
S13	SELECT co_id FROM country WHERE co_name = ?	BuyReq, BuyConf
S14	SELECT count(o_id) FROM orders	BuyConf
S15	SELECT i_id, i_title, a_fname, a_lname FROM item, author WHERE item.i_a_id = author.a_id AND item.i_subject...	NewProd
S16	SELECT i_related1 FROM item where i_id = ?	ShopCart
S17	SELECT i_stock FROM item WHERE i_id = ?	BuyConf
S18	SELECT i.*, a.* FROM author a, item i, author_search a_search WHERE match(a_search.a_lname) against (?) ...	Search
S19	SELECT i.*, a.* FROM item i, item_search i_search, author a WHERE i.i_a_id = a.a_id AND i_search.i_id=i.i_id...	Search
S20	SELECT J.i_id,J.i_thumbnail from item I, item J where (I.i_related1 = J.i_id or I.i_related2 = J.i_id or I.i_related3...	BestSell, Search, Home, NewProd,SrchReq, ShopCart
S21	SELECT max(addr_id) FROM address	BuyConf, BuyReq
S22	SELECT max(c_id) FROM customer	BuyReq
S23	SELECT MAX(o_id)-3333 FROM orders	BestSell
S24	SELECT o_id FROM customer, orders WHERE customer.c_id = orders.o_c_id AND c_uname = ? ORDER BY o_date...	OrdDisp
S25	SELECT ol2.ol_i_id, SUM(ol2.ol_qty) AS sum_ol FROM order_line ol, order_line ol2, $tmp_admin56 t WHERE ...	AdmConf
S26	SELECT orders.*, customer.*, cc_xacts.cx_type, ship.addr_street1 AS ship_addr_street1, ship.addr_street2 AS ...	OrdDisp

Tabela C.3: Consultas e as interações web em que ocorrem

Table C.4: Consultas do Browsing Mix, com uso de cache - primeira série (100k itens, 300 nav.)

	suspensões	invalidações	hits	custo hits	c. médio hit	misses	custo miss	c. médio miss	taxa hits	acessos	max. tuplas
S1	1	54	43	149	3	334	5.571.297	16.681	0,11	377	28
S2	12	0	31	89	3	593	5.530.340	9.326	0,05	624	1.701
S3	13	0	2.113	646.620	306	13	3.270.411	251.570	0,99	2.126	1.200
S4	46	0	8.678	77.200	9	3.173	43.662.314	13.761	0,73	11.851	12.194
S5	1	75	11	99	9	105	3.129.213	29.802	0,09	116	13
S6	14	334	2.535	270.959	107	3.723	1.421.330.338	381.770	0,41	6.258	1.200
S7	12	53	46	125	3	355	1.222.766	3.444	0,11	401	30
S8	0	58	0	0	-	59	863.651	14.638	0,00	59	1
S9	0	43	435	859	2	354	1.010.854	2.856	0,55	789	39
S10	0	43	47	145	3	354	1.254.260	3.543	0,12	401	39
S11	0	41	65	205	3	615	5.952.587	9.679	0,10	680	205
S12	0	34	23	70	3	125	736.896	5.895	0,16	148	13
S13	0	0	52	179	3	11	507.445	46.131	0,83	63	80
S14	0	372	29	29	1	372	991.374	2.665	0,07	401	0
S15	23	55	5.017	534.240	106	1.139	215.199.650	188.937	0,81	6.156	1.200
S16	2	0	5	14	3	508	7.128.317	14.032	0,01	513	1.432
S17	32	0	13	39	3	407	4.860.078	11.941	0,03	420	1.110
S18	11	0	1.600	216.815	136	456	4.513.735	9.899	0,78	2.056	16.918
S19	3	0	1.382	32.776	24	674	9.755.039	14.473	0,67	2.056	14.284
S20	137	0	28.965	1.025.434	35	14.258	203.648.750	14.283	0,67	43.223	367.575
S21	0	58	4	4	1	58	182.458	3.146	0,06	62	0
S22	0	50	0	0	-	50	83.806	1.676	0,00	50	0
S23	0	340	5.917	7.880	1	341	811.258	2.379	0,95	6.258	0
S24	0	88	16	25	2	132	585.616	4.436	0,11	148	4
S25	0	0	0	0	-	58	90.772.017	1.565.035	0,00	58	0
S26	0	77	10	20	2	106	2.749.515	25.939	0,09	116	4
Total:	307	1.775	57.037	2.813.975	49	28.373	2.035.323.985	71.735	0,67	85.410	419.270

	suspensões	invalidações	hits	custo hits	c. médio hit	misses	custo miss	c. médio miss	taxa hits	acessos	max. tuplas
S1	8	105	119	443	4	1.145	9.268.301	8.095	0,09	1.264	83
S2	57	0	510	1.747	3	1.683	17.919.114	10.647	0,23	2.193	5.247
S3	50	0	3.153	1.035.749	328	50	8.697.720	173.954	0,98	3.203	1.200
S4	151	0	6.071	34.607	6	3.847	58.701.574	15.259	0,61	9.918	14.456
S5	6	192	15	105	7	269	4.608.513	17.132	0,05	284	22
S6	39	481	457	42.886	94	2.456	944.830.140	384.703	0,16	2.913	1.200
S7	33	93	55	159	3	574	1.048.740	1.827	0,09	629	43
S8	0	122	0	0	-	126	1.701.693	13.506	0,00	126	1
S9	2	72	648	1.213	2	573	931.490	1.626	0,53	1.221	43
S10	1	72	61	184	3	568	1.188.023	2.092	0,10	629	43
S11	2	68	66	192	3	639	7.029.420	11.001	0,09	705	195
S12	3	71	45	157	3	366	1.246.801	3.407	0,11	411	27
S13	0	0	119	440	4	9	14.991	1.666	0,93	128	89
S14	0	588	41	42	1	588	879.110	1.495	0,07	629	0
S15	41	39	2.216	206.955	93	767	106.974.182	139.471	0,74	2.983	1.200
S16	7	0	18	60	3	820	7.916.440	9.654	0,02	838	2.239
S17	663	0	28	85	3	1.247	9.869.656	7.915	0,02	1.275	1.876
S18	60	0	2.621	112.366	43	653	19.669.192	30.121	0,80	3.274	21.940
S19	42	0	2.291	57.456	25	875	11.678.442	13.347	0,72	3.166	18.867
S20	598	0	27.848	361.615	13	14.914	215.596.240	14.456	0,65	42.762	364.110
S21	0	122	2	3	2	122	238.476	1.955	0,02	124	0
S22	0	91	0	0	-	91	285.034	3.132	0,00	91	0
S23	0	481	2.432	2.730	1	481	617.747	1.284	0,83	2.913	0
S24	0	252	48	48	1	363	1.126.750	3.104	0,12	411	6
S25	0	0	0	0	-	42	57.507.363	1.369.223	0,00	42	0
S26	0	206	13	24	2	271	6.242.582	23.035	0,05	284	6
Total:	1.763	3.055	48.877	1.859.266	38	33.539	1.495.787.734	44.598	0,59	82.416	432.893

Tabela C.5: Consultas do Shopping Mix, com uso de cache - primeira série (100k itens, 300 nav.)

Tabela C.6: Consultas do Ordering Mix, com uso de cache - primeira série (100k itens, 300 nav.)

	suspensões	invalidações	hits	custo hits	c. médio hit	misses	custo miss	c. médio miss	taxa hits	acessos	max. tuplas
S1	93	359	622	2.136	3	6.736	14.896.469	2.211	0,08	7.358	114
S2	127	0	1.691	5.819	3	4.132	8.018.363	1.941	0,29	5.823	14.129
S3	51	0	2.481	756.991	305	51	3.724.001	73.020	0,98	2.532	1.200
S4	161	0	3.630	18.281	5	3.725	5.583.221	1.499	0,49	7.355	13.383
S5	2	105	3	20	7	107	774.433	7.238	0,03	110	9
S6	7	226	6	590	98	270	28.893.391	107.013	0,02	276	400
S7	258	350	504	1.374	3	5.541	3.126.358	564	0,08	6.045	87
S8	0	390	1	1	1	420	3.436.250	8.182	0,00	421	1
S9	50	122	6.671	12.701	2	5.130	3.368.936	657	0,57	11.801	232
S10	75	122	966	3.556	4	5.079	4.373.555	861	0,16	6.045	232
S11	7	69	65	197	3	562	2.941.205	5.233	0,10	627	205
S12	1	53	14	43	3	108	127.809	1.183	0,11	122	7
S13	0	0	437	1.334	3	0	0	-	1,00	437	92
S14	0	5.619	426	410	1	5.619	2.972.834	529	0,07	6.045	0
S15	8	50	76	6.613	87	240	16.717.260	69.655	0,24	316	800
S16	49	0	257	1.287	5	2.950	2.602.987	882	0,08	3.207	9.271
S17	999	0	1.136	3.557	3	4.587	4.279.196	933	0,20	5.723	12.401
S18	60	0	2.007	86.824	43	565	2.952.425	5.226	0,78	2.572	19.760
S19	57	0	1.769	41.553	23	815	3.318.930	4.072	0,68	2.584	16.593
S20	678	0	16.566	182.257	11	13.909	18.487.190	1.329	0,54	30.475	307.980
S21	0	390	16	13	1	390	376.392	965	0,04	406	0
S22	1	148	0	0	-	148	98.179	663	0,00	148	0
S23	0	227	48	40	1	228	82.871	363	0,17	276	0
S24	2	115	5	7	1	117	139.100	1.189	0,04	122	3
S25	0	0	0	0	-	66	22.267.242	337.382	0,00	66	0
S26	1	105	3	6	2	107	1.537.570	14.370	0,03	110	3
Total:	2.687	8.450	39.400	1.125.610	29	61.602	155.096.167	2.518	0,39	101.002	396.902

(a)

Comando	ocorrências	custo total (μs)	custo médio (μs)
I1	58	161.680	2.788
I2	372	1.074.071	2.887
I3	50	85.668	1.713
I4	393	3.876.460	9.864
I5	372	1.167.672	3.139
U1	377	1.095.212	2.905
U2	58	516.929	8.913
U3	420	35.941.976	85.576
Total:	2.100	43.919.668	20.914

(b)

Acertos na cache de comandos SQL	88.088
Falhas na cache de comando SQL	58
Custo total de acerto de comando (μs)	3.034.983
Custo total de falha de comando (μs)	109.576
Custo médio de acerto de comando (μs)	34
Custo médio de falha de comando (μs)	1.889
Eventos de atualização de cache	886
Custo total de atualização de cache (μs)	51.873
Custo médio de atualização de cache (μs)	59

(c)

Tab. dep. global	entradas
ITEM.I_COST	139
ITEM.I_IMAGE	139
ITEM.I_PUB_DATE	139
ITEM.I_RELATED1	139
ITEM.I_RELATED2	139
ITEM.I_RELATED3	139
ITEM.I_RELATED4	139
ITEM.I_RELATED5	139
ITEM.I_THUMBNAIL	139
ITEM.I_STOCK	1.236
CUSTOMER.C_EXPIRATION	642
CUSTOMER.C_LOGIN	642
Total:	3.771

Tabela C.7: Operações de escrita no banco de dados, Browsing Mix - primeira série (100k itens, 300 nav.)

(a)

Comando	ocorrências	custo total (μs)	custo médio (μs)
I1	122	242.952	1.991
I2	588	1.073.170	1.825
I3	91	133.137	1.463
I4	1.208	5.961.585	4.935
I5	588	1.396.048	2.374
U1	1.264	2.594.174	2.052
U2	42	191.841	4.568
U3	1.275	54.969.259	43.113
Total:	5.178	66.562.166	12.855

(b)

Acertos na cache de comandos SQL	91.871
Falhas na cache de comando SQL	42
Custo total de acerto de comando (μs)	2.838.668
Custo total de falha de comando (μs)	60.142
Custo médio de acerto de comando (μs)	31
Custo médio de falha de comando (μs)	1.432
Eventos de atualização de cache	2.026
Custo total de atualização de cache (μs)	140.929
Custo médio de atualização de cache (μs)	70

(c)

Tab. dep. global	entradas
ITEM.I_COST	137
ITEM.I_IMAGE	137
ITEM.I_PUB_DATE	137
ITEM.I_RELATED1	137
ITEM.I_RELATED2	137
ITEM.I_RELATED3	137
ITEM.I_RELATED4	137
ITEM.I_RELATED5	137
ITEM.I_THUMBNAIL	137
ITEM.I_STOCK	3.816
CUSTOMER.C_EXPIRATION	1.234
CUSTOMER.C_LOGIN	1.234
Total:	7.517

Tabela C.8: Operações de escrita no banco de dados, Shopping Mix - primeira série (100k itens, 300 nav.)

(a)

Comando	ocorrências	custo total (μs)	custo médio (μs)
I1	390	274.753	704
I2	5.619	2.766.579	492
I3	148	111.636	754
I4	5.418	4.526.680	835
I5	5.619	3.723.398	663
U1	7.358	4.890.629	665
U2	66	29.052	440
U3	5.723	9.767.988	1.707
Total:	30.341	26.090.715	860

(b)

Acertos na cache de comandos SQL	134.665
Falhas na cache de comando SQL	66
Custo total de acerto de comando (μs)	3.281.387
Custo total de falha de comando (μs)	111.864
Custo médio de acerto de comando (μs)	24
Custo médio de falha de comando (μs)	1.695
Eventos de atualização de cache	13.617
Custo total de atualização de cache (μs)	454.996
Custo médio de atualização de cache (μs)	33

(c)

Tab. dep. global	entradas
ITEM.I_COST	178
ITEM.I_IMAGE	178
ITEM.I_PUB_DATE	178
ITEM.I_RELATED1	178
ITEM.I_RELATED2	178
ITEM.I_RELATED3	178
ITEM.I_RELATED4	178
ITEM.I_RELATED5	178
ITEM.I_THUMBNAIL	178
ITEM.I_STOCK	14.231
CUSTOMER.C_EXPIRATION	1.807
CUSTOMER.C_LOGIN	1.807
Total:	19.447

Tabela C.9: Operações de escrita no banco de dados, Ordering Mix - primeira série (100k itens, 300 nav.)

Table C.10: Consultas do Browsing Mix, com uso de cache - segunda série (1M itens, 50 nav.)

	suspensões	invalidações	hits	custo hits	c. médio hit	misses	custo miss	c. médio miss	taxa hits	acessos	max. tuplas
S1	0	8	16	61	4	51	962.887	18.880	0,24	67	31
S2	0	0	2	6	3	118	616.897	5.228	0,02	120	794
S3	1	0	371	84.512	228	1	847.932	847.932	1,00	372	1.200
S4	0	0	1.503	5.815	4	729	7.010.531	9.617	0,67	2.232	7.433
S5	0	12	1	9	9	16	525.802	32.863	0,06	17	13
S6	2	55	567	54.928	97	641	101.686.574	158.637	0,47	1.208	1.200
S7	1	9	12	28	2	55	48.234	877	0,18	67	29
S8	0	10	0	0	-	10	430.550	43.055	0,00	10	1
S9	0	9	80	154	2	53	76.137	1.437	0,60	133	29
S10	0	9	15	47	3	52	55.163	1.061	0,22	67	29
S11	0	9	15	75	5	110	1.447.129	13.156	0,12	125	48
S12	0	6	4	11	3	18	187.635	10.424	0,18	22	19
S13	0	0	5	21	4	5	2.205	441	0,50	10	52
S14	0	63	4	5	1	63	23.850	379	0,06	67	0
S15	2	8	1.008	93.922	93	202	230.769.675	1.142.424	0,83	1.210	1.200
S16	0	0	0	0	-	99	55.001	556	0,00	99	623
S17	2	0	1	3	3	68	30.437	448	0,01	69	499
S18	0	0	143	4.994	35	227	12.766.536	56.240	0,39	370	15.188
S19	0	0	116	2.530	22	274	24.093.134	87.931	0,30	390	11.331
S20	0	0	479	5.031	11	7.683	25.401.697	3.306	0,06	8.162	298.195
S21	0	10	0	0	-	10	19.237	1.924	0,00	10	0
S22	0	9	0	0	-	9	1.314	146	0,00	9	0
S23	0	55	1.153	1.211	1	55	22.346	406	0,95	1.208	0
S24	0	14	3	4	1	19	189.685	9.983	0,14	22	6
S25	0	0	0	0	-	11	349.428	31.766	0,00	11	0
S26	0	12	1	2	2	16	743.933	46.496	0,06	17	6
Total:	8	298	5.499	253.369	46	10.595	408.363.949	38.543	0,34	16.094	337.926

	suspensões	invalidações	hits	custo hits	c. médio hit	misses	custo miss	c. médio miss	taxa hits	acessos	max. tuplas
S1	0	26	49	186	4	210	1.662.455	7.916	0,19	259	48
S2	0	0	62	217	4	327	647.810	1.981	0,16	389	2.597
S3	6	0	628	150.840	240	6	4.534.393	755.732	0,99	634	1.200
S4	7	0	910	3.212	4	935	6.377.165	6.820	0,49	1.845	8.704
S5	1	35	5	37	7	65	2.250.558	34.624	0,07	70	18
S6	2	96	122	11.218	92	434	68.013.373	156.713	0,22	556	1.050
S7	9	23	17	41	2	121	46.385	383	0,12	138	33
S8	0	29	0	0	-	30	839.359	27.979	0,00	30	1
S9	0	17	154	271	2	112	52.403	468	0,58	266	33
S10	0	17	26	80	3	112	40.594	362	0,19	138	33
S11	0	14	8	22	3	90	1.103.317	12.259	0,08	98	66
S12	0	15	16	62	4	64	193.888	3.030	0,20	80	27
S13	0	0	22	69	3	11	3.454	314	0,67	33	66
S14	0	129	9	8	1	129	43.634	338	0,07	138	0
S15	7	3	456	38.719	85	79	63.582.963	804.848	0,85	535	1.199
S16	0	0	0	0	-	151	58.836	390	0,00	151	1.009
S17	130	0	4	9	2	257	1.376.972	5.358	0,02	261	913
S18	8	0	274	10.238	37	343	16.321.208	47.584	0,44	617	22.987
S19	4	0	173	3.985	23	477	35.629.680	74.695	0,27	650	16.637
S20	47	0	419	4.518	11	7.757	22.003.605	2.837	0,05	8.176	293.780
S21	0	29	3	3	1	29	4.911	169	0,09	32	0
S22	0	23	0	0	-	23	13.531	588	0,00	23	0
S23	0	95	461	437	1	95	26.684	281	0,83	556	0
S24	0	39	9	19	2	71	200.301	2.821	0,11	80	6
S25	0	0	0	0	-	3	57.281	19.094	0,00	3	0
S26	0	36	7	13	2	63	1.873.671	29.741	0,10	70	6
Total:	221	626	3.834	224.204	58	11.994	226.958.431	18.923	0,24	15.828	350.413

Tabela C.11: Consultas do Shopping Mix, com uso de cache - segunda série (1M itens, 50 nav.)

	suspensões	invalidações	hits	custo hits	c. médio hit	misses	custo miss	c. médio miss	taxa hits	acessos	max. tuplas
S1	5	59	314	1.213	4	1.079	2.524.253	2.339	0,23	1.393	52
S2	2	0	130	435	3	963	8.327.693	8.648	0,12	1.093	7.114
S3	1	0	511	123.062	241	1	665.179	665.179	1,00	512	1.200
S4	3	0	529	2.076	4	887	6.291.288	7.093	0,37	1.416	7.543
S5	1	23	1	11	11	25	164.460	6.578	0,04	26	5
S6	0	30	1	93	93	36	7.947.837	220.773	0,03	37	150
S7	44	63	218	558	3	912	309.844	340	0,19	1.130	49
S8	0	65	0	0	-	70	565.672	8.081	0,00	70	1
S9	0	22	1.704	2.986	2	507	133.920	264	0,77	2.211	66
S10	2	22	621	2.335	4	509	163.432	321	0,55	1.130	66
S11	0	15	10	33	3	104	1.262.295	12.137	0,09	114	33
S12	0	9	3	10	3	25	43.283	1.731	0,11	28	12
S13	0	0	73	232	3	0	0	-	1,00	73	92
S14	0	1.102	28	18	1	1.102	294.931	268	0,02	1.130	0
S15	0	8	10	891	89	41	36.019.407	878.522	0,20	51	649
S16	0	0	4	16	4	584	869.737	1.489	0,01	588	4.087
S17	148	0	79	215	3	995	3.048.754	3.064	0,07	1.074	5.965
S18	3	0	184	6.825	37	297	11.448.717	38.548	0,38	481	19.463
S19	1	0	145	3.084	21	329	21.634.825	65.759	0,31	474	13.762
S20	13	0	210	2.326	11	5.498	9.057.455	1.647	0,04	5.708	210.135
S21	0	65	3	3	1	65	32.497	500	0,04	68	0
S22	2	24	0	0	-	24	5.387	224	0,00	24	0
S23	0	30	7	9	1	30	5.564	185	0,19	37	0
S24	1	25	1	2	2	27	42.202	1.563	0,04	28	2
S25	0	0	0	0	-	9	66.359	7.373	0,00	9	0
S26	0	23	1	2	2	25	718.522	28.741	0,04	26	2
Total:	226	1.585	4.787	146.435	31	14.144	111.643.513	7.893	0,25	18.931	270.448

Tabela C.12: Consultas do Ordering Mix, com uso de cache - segunda série (1M itens, 50 nav.)

(a)

Comando	ocorrências	custo total (μs)	custo médio (μs)
I1	10	17.145	1.715
I2	63	32.816	521
I3	9	21.850	2.428
I4	65	1.023.643	15.748
I5	63	343.567	5.453
U1	67	75.839	1.132
U2	11	849.875	77.261
U3	69	5.748.026	83.305
Total:	357	8.112.761	22.725

(b)

Acertos na cache de comandos SQL	16.560
Falhas na cache de comando SQL	11
Custo total de acerto de comando (μs)	356.919
Custo total de falha de comando (μs)	9.099
Custo médio de acerto de comando (μs)	22
Custo médio de falha de comando (μs)	827
Eventos de atualização de cache	154
Custo total de atualização de cache (μs)	7.187
Custo médio de atualização de cache (μs)	47

(c)

Tab. dep. global	entradas
ITEM.I_COST	68
ITEM.I_IMAGE	68
ITEM.I_PUB_DATE	68
ITEM.I_RELATED1	68
ITEM.I_RELATED2	68
ITEM.I_RELATED3	68
ITEM.I_RELATED4	68
ITEM.I_RELATED5	68
ITEM.I_THUMBNAIL	68
ITEM.I_STOCK	557
CUSTOMER.C_EXPIRATION	284
CUSTOMER.C_LOGIN	284
Total:	1.737

Tabela C.13: Operações de escrita no banco de dados, Browsing Mix - segunda série (1M itens, 50 nav.)

(a)

Comando	ocorrências	custo total (μs)	custo médio (μs)
I1	29	11.509	397
I2	129	45.661	354
I3	23	20.654	898
I4	241	1.804.868	7.489
I5	129	244.255	1.893
U1	259	83.685	323
U2	3	843	281
U3	261	8.071.228	30.924
Total:	1.074	10.282.703	9.574

(b)

Acertos na cache de comandos SQL	17.762
Falhas na cache de comando SQL	3
Custo total de acerto de comando (μs)	447.150
Custo total de falha de comando (μs)	2.435
Custo médio de acerto de comando (μs)	25
Custo médio de falha de comando (μs)	812
Eventos de atualização de cache	423
Custo total de atualização de cache (μs)	23.341
Custo médio de atualização de cache (μs)	55

(c)

Tab. dep. global	entradas
ITEM.I_COST	69
ITEM.I_IMAGE	69
ITEM.I_PUB_DATE	69
ITEM.I_RELATED1	69
ITEM.I_RELATED2	69
ITEM.I_RELATED3	69
ITEM.I_RELATED4	69
ITEM.I_RELATED5	69
ITEM.I_THUMBNAIL	69
ITEM.I_STOCK	1.864
CUSTOMER.C_EXPIRATION	500
CUSTOMER.C_LOGIN	500
Total:	3.485

Tabela C.14: Operações de escrita no banco de dados, Shopping Mix - segunda série (1M itens, 50 nav.)

(a)

Comando	ocorrências	custo total (μs)	custo médio (μs)
I1	65	19.736	304
I2	1.102	299.073	271
I3	24	9.904	413
I4	1.055	409.795	388
I5	1.102	515.706	468
U1	1.393	380.890	273
U2	9	2.530	281
U3	1.074	8.673.151	8.076
Total:	5.824	10.310.785	1.770

(b)

Acertos na cache de comandos SQL	25.252
Falhas na cache de comando SQL	9
Custo total de acerto de comando (μs)	423.687
Custo total de falha de comando (μs)	7.080
Custo médio de acerto de comando (μs)	17
Custo médio de falha de comando (μs)	787
Eventos de atualização de cache	2.556
Custo total de atualização de cache (μs)	61.869
Custo médio de atualização de cache (μs)	24

(c)

Tab. dep. global	entradas
ITEM.I_COST	77
ITEM.I_IMAGE	77
ITEM.I_PUB_DATE	77
ITEM.I_RELATED1	77
ITEM.I_RELATED2	77
ITEM.I_RELATED3	77
ITEM.I_RELATED4	77
ITEM.I_RELATED5	77
ITEM.I_THUMBNAIL	77
ITEM.I_STOCK	7.081
CUSTOMER.C_EXPIRATION	655
CUSTOMER.C_LOGIN	655
Total:	9.084

Tabela C.15: Operações de escrita no banco de dados, Ordering Mix - segunda série (1M itens, 50 nav.)

Referências Bibliográficas

[1] Cardellini, V., Casalicchio, E., Colajanni, M., et al., *The State of the Art in Locally Distributed Web-server Systems.* IBM Research Report, RC22209 (W0110-048) October 16, 2001

[2] Titchkosky, L., Arlittz, M., Williamson, C., "A Performance Comparison of Dynamic Web Technologies." *ACM SIGMETRICS Performance Evaluation Review*, Volume 31 , Issue 3 (December 2003)

[3] Q. Zhang, A. Riska, E. Riedel, E. Smirni, "Bottlenecks and their Performance Implications in E-Commerce Systems", In: *Proceedings of Ninth International Workshop on Web Content Caching and Distribution (WCW2004)*, pp. 273-282, Beijing, China, Oct. 2004.

[4] "MySQL Cluster" - http://www.mysql.com/products/database/cluster/

[5] Kemme, B., Alonso, G, "Don't Be Lazy, Be Consistent: Postgres-R, a New Way to Implement Database Replication", In: *Proceedings of the 26th International Conference on Very Large Databases*, Cairo, Egypt, September 2000.

[6] "PGCluster - The Multi-master and Synchronous Replication System for PostgreSQL" - http://pgcluster.projects.postgresql.org/

[7] Cavalleri, M., Prudentino, .R, Pozzoli, U. et. al. "A Set of tools for building PostgreSQL distributed databases in biomedical environment", *22th Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, Jul 2000.

[8] "Projeto PARGRES - Processamento Paralelo de Consultas no Sistema de Banco de Dados PostgreSQL" http://pargres.nacad.ufrj.br/

[9] Dias, S. B., *Técnicas Baseadas em Memória Compartilhada e Passagem de Mensagens para Servidores de Aplicações de Comércio Eletrônico com Qualidade de Serviço.* Tese de M.Sc., Engenharia de Sistemas e Computação, COPPE/UFRJ, 2003.

[10] Apache Software Foundation, “Java Caching System” - http://jakarta.apache.org/turbine/jcs/, accessed January 5th, 2005.

[11] Watkinson, J., “SwarmCache - Cluster-aware Caching for Java” - http://swarmcache.sourceforge.net/, accessed January 5th, 2005.

[12] Ban, B., Wang, B., Gliebe, H., “JBoss Cache” - http://www.jboss.org/products/jbosscache, accessed January 5th, 2005.

[13] Cecchet, E., Marguerite, J., Zwaenepoel, W., “C-JDBC: Flexible Database Clustering Middleware”. *USENIX 2004 Annual Technical Conference*, pp. 9-18

[14] Luo, Q., Krishnamurthy, S., Mohan, C., Pirahesh, H., Woo, H., Lindsay, B., G., Naughton, J., F., "Middle-Tier Database Caching for e-Business", In: *Proceedings of the 2002 ACM SIGMOD international conference on Management of data*, June 4-6, 2002.

[15] Isocra Ltd., “Livestore - Distributed J2EE caching” - http://www.isocra.com/livestore, accessed January 5th, 2005.

[16] Walther, S., “Improved Caching in ASP.NET 2.0” - MSDN Library, http://msdn.microsoft.com/library/en-us/dnvs05/html/cachingnt2.asp, accessed January 5th, 2005.

[17] Cuenca-Acuna, F. M., Nguyen, T. D., “Cooperative Caching Middleware for Cluster-Based Servers”, In: *Proceedings of the 10th IEEE International Symposium on High Performance Distributed Computing (HPDC)*, August 2001.

[18] Holmedahl, V., Smith, B., Yang, T., “Cooperative Caching of Dynamic Content on a Distributed Web Server”, In: *Proceedings of the Seventh IEEE International Symposium on High Performance Distributed Computing*, July 28 - 31, 1998

[19] Demichiel, L. G., “Enterprise Java Beans Specification, Version 2.1”, November 12, 2003, http://java.sun.com/products/ejb/

[20] Sun Microsystems, Inc., “Java 2 Platform, Enterprise Edition Specification, v.1.4“, Final Release November 24, 2003, http://java.sun.com/j2ee.

[21] Degenaro, L., Iyengar, A., Lipkind, I., Rouvellou, I.,“A Middleware System Which Intelligently Caches Query Results”, *IFIP/ACM International Conference on Distributed systems platforms*, pp.24-44, 2000

[22] Ullman, J. D., Widom, J., *A First Course in Database Systems*, Prentice Hall, 1997.

[23] The Apache Software Foundation, "Apache HTTP Server Project" - http://httpd.apache.org/

[24] The Apache Software Foundation, "Apache Jakarta Tomcat" - http://jakarta.apache.org/tomcat/

[25] BEA Systems, Inc, "Bea Weblogic Platform"- http://www.bea.com/products/weblogic/platform

[26] IBM Corporation ,"WebSphere Application Server" - http://www.ibm.com/software/webservers/appserv/was/

[27] JBoss Inc., "JBoss Application Server" - http://www.jboss.org/docs/index

[28] Garcia-Molina, H., Ullman, J. D., Widow, J., *Database System Implementation.* Prentice Hall (2000).

[29] Henessy, Patterson, "Computer Achitecture, A Quantitative Approach, 3rd. edition"

[30] MySQL AB, *MySQL Administrator's Guide*, MySQL Press, 2005

[31] Greenwald, R., Stackoviak, R., Stern, J., *Oracle Essentials - third edition*, O'Reilly - February 2004

[32] Wiesmann, M., Pedone, F., Schiper, A., Kemme, B., Alonso, G., "Database Replication Techniques: a Three Parameter Classification" - In: *Proceedings of 19th IEEE Symposium on Reliable Distributed Systems (SRDS2000)*

[33] Ceri, S., Houtsma, M. A. W., Keller, A. M., Samarati, P., *A Classification of Update Methods for Replicated Databases*, Technical Report: CS-TR-91-1392, 1991

[34] Menascé, D., A., Almeida, V., F. , Fonseca, R., et. al., "A Methodology for Workload Characterization of E-commerce Sites", In: *Proceedings of the 1st ACM conference on Electronic commerce*, 1999

[35] Fischer, M., Ellis, J., Bruce, J., *JDBC API Tutorial and Reference*, Third Edition. Addison-Wesley (2003).

[36] Menascé, D. A., Almeida, V. F. , Fonseca, R., *Scaling for E-Business - Technologies, Models, Performance, and Capacity Planning* - Prentice Hall PTR, 2000

[37] Wang, Q., Makaroff, D.,Edwards, H. K., Thompson, R., "Workload characterization for an E-commerce web site" - In: *IBM Centre for Advanced Studies Conference, proceedings of the 2003 conference of the Centre for Advanced Studies on Collaborative research*, Toronto, Ontario, Canada pp. 313 - 327 , 2003

[38] Transaction Processing Performance Council (TPC), "TPC Benchmark W (Web Commerce) Specification Version 1.8" - http://www.tpc.org, Feb.19, 2002

[39] Nanda, N., "JDBC drivers in the wild - Learn how to deploy, use, and benchmark JDBC driver types 1, 2, 3, and 4" - JavaWorld, July-2000 http://www.javaworld.com/javaworld/jw-07-2000/jw-0707-jdbc.html

[40] Arlitt, M., Krishnamurthy, D., Rolia, J.,"Characterizing the scalability of a Large Web-Based Shopping System" - *ACM Transactions on Internet Technology*, Volume 1, Number 1, August, 2001

[41] "Java Compiler Compiler [tm] (JavaCC [tm]) - The Java Parser Generator" - https://javacc.dev.java.net/

[42] AHO, A. V., SETHI, R., ULLMAN, J. D., *Compiladores, Princípios, Técnicas e Ferramentas* - LTC Editora, 1995

[43] Grune, D., Bal, E. H., Jacobs, C. J. H., Langendoen, K. G., *Projeto Moderno de Compiladores - Implementação e Aplicações* - Editora Campus, 2001

[44] MySQL AB, "MySQL Reference Manual", http://www.mysql.com/documentation/

[45] Godard, S., "Sysstat", http://perso.wanadoo.fr/sebastien.godard/

[46] UW-Madison Computer Architecture Group, "Java TPC-W Implementation Distribution", http://www.ece.wisc.edu/~pharm/

[47] ObjectWeb - Open-Source Middleware - http://consortium.objectweb.org/

[48] Roubtsov, V. "Java Tip 130: Do you know your data size?", JavaWorld, August 16, 2002, http://www.javaworld.com/javaworld/javatips/jw-javatip130.html

[49] Amza, C., Cecchet, E., Chanda, A., et al, "Specification and Implementation of Dynamic Web Site Benchmarks", IEEE 5th Annual Workshop on Workload Characterization (WWC-5), Austin, TX, USA, November 2002.

[50] Dar, S., Franklin, M. J., Jónsson, B. T., "Semantic Data Caching and Replacement", In: *Proceedings on th 22nd VLDB Conference,* Mumbai(Bombay), India, September, 1996

[51] Soundararajan, G., Amza, C., "Using Semantic Information to Improve Transparent Query Caching for Dynamic Content Web Sites", Data Engineering Issues in E-Commerce (DEEC 2005) - 21st International Conference on Data Engineering (ICDE 2005).

[52] Deshpande, P., Ramasamy, K., Shukla A., et al., "Caching Multidimensional Queries Using Chunks" In: *Proceedings ACM SIGMOD International Conference on Management of Data*, June 2-4, 1998, Seattle, Washington, USA, pp. 259-270