UNIVERSIDADE DE BRASÍLIA
FACULDADE DE TECNOLOGIA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

BANCO DE DADOS EM MEMÓRIA PRINCIPAL, UM ESTUDO DE CASO: ORACLE TIMESTEN SOLUÇÃO DE ALTO DESEMPENHO

RENATO COSTA PEREIRA

ORIENTADOR: GEORGES DANIEL AMVAME NZE

DISSERTAÇÃO DE MESTRADO EM ENGENHARIA ELÉTRICA

PUBLICAÇÃO: PPGENE.DM - 065/10
BRASÍLIA/DF: AGOSTO – 2010

UNIVERSIDADE DE BRASÍLIA
FACULDADE DE TECNOLOGIA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

"BANCO DE DADOS EM MEMÓRIA PRINCIPAL, UM ESTUDO DE CASO: ORACLE TIMESTEN SOLUÇÃO DE ALTO DESEMPENHO"

RENATO COSTA PEREIRA

DISSERTAÇÃO DE MESTRADO PROFISSIONALIZANTE SUBMETIDA AO DEPARTAMENTO DE ENGENHARIA ELÉTRICA DA FACULDADE DE TECNOLOGIA DA UNIVERSIDADE DE BRASÍLIA, COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE.

APROVADA POR:

GEORGES DANIEL AMVAME NZÉ, Dr., FGA/UNB
(ORIENTADOR)

FLÁVIO ELIAS GOMES DE DEUS, Dr., ENE/UNB
(EXAMINADOR INTERNO)

JACIR LUIZ BORDIM, Dr., CIC/UNB
(EXAMINADOR EXTERNO)

BRASÍLIA, 26 DE JULHO DE 2010.

FICHA CATALOGRÁFICA

PEREIRA, RENATO COSTA

Banco de Dados em Memória Principal, um Estudo de Caso: Oracle TimesTen Solução de Alto Desempenho [Distrito Federal] 2010.

xiii, 61p., 210 x 297 mm (ENE/FT/UnB, Mestre, Dissertação de Mestrado – Universidade de Brasília, Faculdade de Tecnologia.

Departamento de Engenharia Elétrica

1. Banco de Dados
2. Desempenho
3. IMDB
4. Cache de Banco de Dados
I. ENE/FT/UnB
II. Título (série)

REFERÊNCIA BIBLIOGRÁFICA

PEREIRA, R. C. (2010). Banco de Dados em Memória Principal, um Estudo de Caso: Oracle TimesTen Solução de Alto Desempenho. Dissertação de Mestrado em Engenharia Elétrica, Publicação PPGENE.DM-065/10, Departamento de Engenharia Elétrica, Universidade de Brasília, Brasília, DF, 61p.

CESSÃO DE DIREITOS

AUTOR: Renato Costa Pereira.

TÍTULO: Banco de Dados em Memória Principal, um Estudo de Caso: Oracle TimesTen Solução de Alto Desempenho.

GRAU: Mestre ANO: 2010

Renato Costa Pereira

Super Quadra Sul (SQS) 113 Bloco J Apt. 501

70.376-100 Brasília – DF – Brasil.

Agradecimentos

A minha mãe pelo incentivo ao desenvolvimento da minha carreira na área de tecnologia da informação, que me proporcionou realizar as experiências aqui relatadas.

Agradeço a Viviane, pelo imenso apoio, compreensão e conforto carinhosamente oferecidos ao longo da elaboração desse trabalho.

Ao meu orientador, professor Georges Amvame Nze, pela aposta que fez ao me aceitar como aluno, e também pela colaboração e boa vontade dispendidas nesse trabalho.

Dedico este trabalho a toda minha família.

RESUMO

BANCO DE DADOS EM MEMÓRIA PRINCIPAL, UM ESTUDO DE CASO: ORACLE TIMESTEN SOLUÇÃO DE ALTO DESEMPENHO.

Autor: Renato Costa Pereira

Orientador: Georges Daniel Amvame Nze

Programa de Pós-graduação em Engenharia Elétrica

Brasília, agosto de 2010.

Na era do conhecimento, basicamente todas as informações das empresas utilizam-se de uma memória organizacional normalmente representada pelos bancos de dados. Eles podem, obviamente, armazenar dados relevantes aos serviços e negócios da organização. Disponibilizar essas informações aos usuários e clientes no menor prazo e custo acessível tornou-se uma exigência e uma necessidade imprescindível dentro do contexto de negócios. Mantê-las disponíveis, bem como ajustar parâmetros e configurações do programa de gerenciamento de banco de dados, para melhorar o tempo de resposta às consultas feitas nessas bases, são algumas das atribuições do Administrador de Banco de Dados (DBA). Visando cumprir essas e outras tarefas em tempo hábil, é possível que haja a necessidade de outras opções de programas com características de tempo real ou alguma outra funcionalidade que ofereça o desempenho desejado. Esse trabalho tem o propósito de elaborar um estudo sobre bancos de dados em memória, objetivando identificar os ganhos de desempenho que podem ser obtidos em relação às soluções tradicionais residentes em disco. Por meio de um estudo específico, serão realizados testes e análises para identificar a real rapidez do tempo de resposta dessa tecnologia.

ABSTRACT

IN-MEMORY DATABASE, A CASE STUDY: ORACLE TIMESTEN HIGH PERFORMANCE SOLUTION.

Author: Renato Costa Pereira

Supervisor: Georges Daniel Amvame Nze

Programa de Pós-graduação em Engenharia Elétrica

Brasília, month of august 2010.

During the knowledge era, all companies information essentially use an organizational memory usually represented by the databases. They can, evidently, store relevant data related to business and services of the organization. Providing this information to users and customers quickly and affordably has become a requirement and an absolute necessity within the business context. Keeping them available, as well as adjusting parameters and settings of the database management software to improve time response to queries in databases, are some of the responsibilities of the Database Administrator (DBA). In order to meet those tasks and others in a timely manner there may possibly be the need for other software options with real-time features or some other features that offers the desired performance. This work aims to conduct a study on in-memory databases in order to identify the performance gains' that can be achieved when compared to traditional solutions residing on disk. Through a specific study, tests and analysis will be conducted to identify the real time response of this technology.

Sumário

LISTA DE TABELAS

LISTA DE FIGURAS

LISTA DE SÍMBOLOS, NOMENCLATURAS E ABREVIAÇÕES

ACID – Atomicidade, consistência, isolamento e durabilidade

AMD – *Advanced Micro Devices*

API – *Application Programming Interface*

BDO – Base de Dados Operacional

B-TREE – *Balanced Tree*

CPD – Centro de Processamento de Dados

CHU – *Charge Units*

CIO – *Chief Information Officer*

CISC – *Complex Instruction Set Computer*

CDR – *Call Detail Record*

DBA – *Database Administrator*

DML – *Data Manipulation Language*

DSN – *Data Source Name*

DRDB – *Disk Resident Database Management*

E/S – Entrada e Saída

EEPROM – *Eletrically-Erasable Programmable Read-Only Memory*

HP – *Hewlett-Packard*

IMDB – *In-memory Database System*

IP – *Internet Protocol*

JDBC – *Java Database Conectivity*

LRU – *Least Recently Used*

NVRAM – *Non-Volatile Random Access*

ODBC – *Open Database Connectivity*

OLTP – *Online Transaction Processing*

OTN – *Oracle Technology Network*

RAM – *Random-Access Memory*

SAN – *Storage Area Network*

SGA – *System Global Area*

SGBD – Sistema de Gerenciamento de Banco de Dados

SGBDR – Sistema de Gerenciamento de Banco de Dados Relacional

SLA – *Service Level Agreement*

SNMP – *Simple Network Management Protocol*

SQL – *Structured Query Language*

TI – Tecnologia da Informação

UCP – Unidade Central de Processamento

VIO – *Virtual I/O*

XML – *Extensible Markup Language*

1 INTRODUÇÃO

1.1 PERSPECTIVA GERAL

Os sistemas de gerenciamento de bancos de dados relacionais (SGBDR) – categoria composta por estruturas apresentadas na forma de tabelas, com possíveis relações entre colunas – são atualmente usados em quase todos os ambientes de computação, visando criar, organizar e manter as bases de informações que suportam instituições privadas ou governamentais, garantindo a segurança e persistência da informação durante o tempo que for necessário. Sua ampla utilização passou a ser comum com o surgimento de poderosas linguagens de consulta, que escondem as peculiaridades de acesso aos dados, como a *Structured Query Language* (SQL) (MANEGOLD, 2002), permitindo tanto usuários finais quanto desenvolvedores acessarem de forma simples e fácil os dados de que necessitam.

O modelo de banco de dados relacional foi desenvolvido principalmente por E. F. Codd (CODD, 1970), no início dos anos 70 e, desde então, diversas pesquisas buscam elevar o desempenho desses sistemas, que está relacionado ao tempo de disponibilidade, ao tempo de resposta de uma solicitação e ao número de transações por segundo. É o SGBDR que determina a facilidade e rapidez com que novas aplicações poderão ser desenvolvidas, e em responder às novas oportunidades de negócio.

Algumas pesquisas apontam o acesso de leitura ou gravação (E/S) aos meios de armazenamento secundários (discos rígidos) como o principal responsável pelos problemas de desempenho do sistema. O problema ocorre devido à latência da transferência dos dados dos discos rígidos para a memória principal e seu posterior acesso pela unidade central de processamento (UCP). Foram desenvolvidas várias técnicas para evitar o acesso, de alto custo, aos discos rígidos; entre elas estão:

- A criação de algoritmos sofisticados de classificação que exploram a quantidade de memória disponível;
- Utilização de estruturas auxiliares de acesso aos dados (índices);
- Particionamento de segmentos de dados, visando diminuir o conjunto de pesquisa com base em um critério conhecido (normalmente o tempo);
- Armazenamento dos dados lidos em áreas intermediárias compartilhadas de memória (*cache*), valendo-se de algoritmos que mantêm nessa área de memória

o dado mais provável de ser acessado: aquele que foi recentemente lido ou algum dado próximo a ele.

A despeito de todos os avanços tecnológicos que ocorrem na indústria de equipamentos de computação, a organização clássica de sistemas computacionais, conhecida como Arquitetura de *Von Neumann*, vem-se mantendo a mesma desde a primeira geração de computadores. Essa arquitetura apresenta severas limitações quando manipula grandes volumes de dados em bancos de dados tradicionais, residentes em disco (*Disk Resident Database Management* – DRDB) ou quando precisa de baixíssimo tempo de resposta em uma transação[1] (KHAN, PAUL, *et al.*, 1999). Esse requisito de desempenho, entre outros motivos, fez emergir as tecnologias de bancos de dados que permanecem completamente residentes em memória principal (*in-memory database system* – IMDB) (PISHARATH, CHOUDHARY e KANDEMIR, 2004), apesar de várias implementações de IMDB utilizarem o disco rígido para realizar cópias de segurança (*backup*).

Ambas as tecnologias necessitam do disco rígido; no entanto, o desempenho superior do IMDB ocorre devido à sua capacidade de possuir acesso aleatório a porções de dados com tamanhos variados, ao contrário do sistema de armazenamento em disco rígido, onde os dados são recuperados sequencialmente em blocos de tamanho fixo. Além disso, os dispositivos de memória são endereçados diretamente pelo processador e possuem latência de velocidade de acesso, com ordens de grandeza bem menores que os discos rígidos.

Nos últimos anos os bancos de dados passaram a armazenar grandes volumes de informação, centenas de *gigabytes*, dificultando, dessa forma, a criação de bases de dados residentes inteiramente na memória. Diante desse cenário, uma das alternativas que possibilitam obter alto desempenho com o IMDB é sua utilização de forma complementar a uma solução tradicional; ou seja, apenas alguns objetos da base de dados tradicional são replicados na base em memória para uma aplicação específica.

[1] Transação: é uma sequência de operações que são tratadas como um bloco único e indivisível, visando garantir sua total execução ou, em caso de falha de alguma operação, que seja completamente desfeita. Além disso, é utilizada para garantir o isolamento entre acessos concorrentes na mesma base de dados.

1.2 MOTIVAÇÃO E RELEVÂNCIA DO ESTUDO

Nos bancos de dados em memória, todos os dados são armazenados na memória de acesso aleatório (*Random-Access Memory* – RAM) primária, a fim de proporcionar acesso muito mais rápido que os bancos de dados tradicionais residentes em discos. O acesso aos dados em discos é frequentemente a operação mais lenta de um sistema de computação, especialmente quando a aplicação faz uso intenso de dados armazenados, como nos bancos de dados. Através da remoção do acesso de leitura ou gravação aos meios de armazenamento em discos da arquitetura da solução, a eficiência e a vazão do sistema podem aumentar consideravelmente.

Devido aos custos da memória principal, as soluções tradicionais de bancos de dados residentes em discos foram projetadas para trabalhar usando uma combinação de dados residentes em discos e de um *cache* em memória principal para um subconjunto de dados, como forma de minimizar os atrasos causados pela espera de dados a serem lidos dos discos. Com a evolução da tecnologia e aumento da densidade de *chips*, a memória principal tornou-se grande o suficiente para manter o banco de dados inteiro em memória, abrindo as portas para o surgimento dos IMDBs, como um tipo diferente de sistema de banco de dados (REID, 2009).

Grande parte das soluções de bancos de dados em memória podem ser utilizadas como bases de dados completamente independentes, como um *cache* conectado a um banco de dados tradicional ou em modo de replicação, quando alguns de seus objetos estão replicados em outras bases de dados em diferentes servidores. Essa diversidade torna atraente a utilização de soluções de bancos de dados em memória em diversos cenários, motivando o estudo comparativo entre as soluções, em memória e tradicionais, em diferentes arquiteturas.

Tal comparação se torna possível devido à possibilidade dos DRDBs poderem ser executados integralmente na memória principal e continuarem totalmente funcionais; entretanto, suas estruturas de armazenamento foram otimizadas especificamente para usar o disco para armazenar dados. A fim de tirar vantagens das características adicionadas por residir primariamente na memória principal, novos sistemas foram concebidos. Não somente o tempo de acesso é significativamente menor nos sistemas residentes em

memória principal, mas eles não precisam ser orientados a blocos porque não há nenhum custo por acesso e o custo do acesso sequencial é insignificante. Quase todos os aspectos de um sistema de banco de dados podem ser afetados por essas diferenças. Isso motiva a investigação sobre o melhor projeto para utilizar a tecnologia de um IMDB para maximizar seu desempenho e garantir um serviço de qualidade.

Apesar dos SGBDRs terem se tornado vitais nas organizações, as grandes empresas de telecomunicações procuram se precaver quanto à utilização de novas tecnologias. Geralmente, nos sistemas de missão crítica, tais como, faturamento e tarifação, arrecadação, mediação e relacionamento com clientes, opta-se por cenários conservadores, utilizando servidores com vasta capacidade de processamento e SGBDRs comerciais com suporte 24 horas por dia, durante todo o ano. Atualmente os IMDBs são também fornecidos por grandes empresas e possuem quase todas as funcionalidades e garantias dos DRDBs, tais como, suporte ao SQL padrão ANSI, independência de transações, entre outros. Dessa forma, seu emprego visa agilizar processos antes realizados com soluções desenvolvidas quando a realidade da infraestrutura era outra.

1.3 OBJETIVO GERAL

Essa dissertação pretende estudar um banco de dados em memória e suas diferenças em relação aos bancos de dados tradicionais residentes em disco. Será utilizado o *TimesTen* (ORACLE, 2009), um sistema de gerenciamento de banco de dados em memória, comparando seu desempenho de processamento com um banco de dados tradicional residente em disco, ambos comercializados pela *Oracle*. Com a premissa de que todos os dados da base estão na memória RAM, característica típica do desenho assumido nas soluções IMDB, espera-se constatar que seu desempenho seja superior às soluções DRDB.

Esse trabalho também tem por finalidade prover material técnico que auxilie empresas de telecomunicações na adoção da tecnologia de bancos de dados em memória, servindo como referência para sua utilização, ou não, em sistemas críticos que necessitem de alto desempenho da camada de banco de dados.

1.3.1 OBJETIVOS ESPECÍFICOS

Para realizar esse estudo comparativo, será necessário criar um banco de dados em memória e um banco de dados convencional, residente em disco, em equipamentos diferentes com as mesmas configurações de UCP, memória, interfaces de rede e discos rígidos, garantindo dessa forma que os produtos testados, embora utilizem os recursos da máquina de forma diferente, terão as mesmas condições de execução dos testes.

Em seguida, será necessário realizar transações semelhantes de leitura e gravação nas bases de dados criadas, visando à coleta de estatísticas (quantidade de usuários, tempo decorrido, etc.) de cada uma dessas transações para posterior comparação.

Para as coletas de estatísticas, tanto da base IMDB quanto da base DRBD, será utilizada a ferramenta *Swingbench*, produto que gera uma massa de dados aleatória em bancos de dados relacionais, e realiza transações pré-configuradas através da linguagem SQL.

1.4 CONTEXTUALIZAÇÃO

De acordo com Garcia-Molina et al (1992), é razoável que alguns bancos de dados sejam mantidos completamente na memória principal do computador. Essa abordagem depende da aplicação, pois, em alguns casos, o banco de dados possui um tamanho limitado ou seu crescimento é menor que a taxa de crescimento da capacidade de memória. Em algumas aplicações de tempo real[2], o dado deve existir na memória principal, visando garantir as restrições desse tipo de aplicação; dessa forma, o banco de dados deve ser menor que a quantidade de memória disponível. Também, segundo os autores (1992), há casos em que claramente a base de dados não caberá na memória e, nesses casos, o DRDB continuará a ser importante. No entanto, segundo ele, mesmo nas aplicações muito grandes é comum encontrar diferentes tipos de dados: *dados quentes* que são frequentemente acessados, geralmente com baixos volumes e requisitos de tempo rigorosos, e, *dados frios*, mais volumosos, porém acessados raramente. Nesse caso é possível particionar os dados em uma ou mais bases de dados lógicas, e armazenar as bases com *dados quentes* na memória RAM.

[2] Temo Real é uma expressão que se refere a sistemas em que o tempo de execução de uma determinada tarefa é rígido.

Para Miyazaki (2005), o IMDB tem vantagens tanto no projeto físico da base de dados, quanto no desempenho em relação ao DRDB. Eles podem usar uma forma mais flexível de estrutura de dados para armazenar as informações na memória principal que os bancos de dados convencionais, que precisam considerar as propriedades do disco. Outra vantagem é a possibilidade de objetos grandes poderem compartilhar atributos que possuem o mesmo valor, visando redução do espaço de memória e, por consequência, maior agilidade na verificação entre dois objetos grandes, através da comparação de ponteiros. Além disso, sua estrutura de dados tem outra vantagem devido ao tamanho de cada tupla também ser fixa, possibilitando a pesquisa de um atributo ser equivalente à busca correspondente de uma palavra na memória em um passo fixo.

Nesse, e no próximo parágrafo, segundo Bitton (1986), no desenho de bancos de dados convencionais, a organização de arquivos, métodos de acesso e gerenciamento de memória são desenhados para reduzir o número de acessos a disco. Pode-se aumentar o desempenho através da utilização de grandes áreas intermediárias de memória (*cache*), onde os dados são pré-lidos ou aqueles que foram frequentemente acessados podem ser mantidos como índices ou *hash tables*. Entretanto, há aplicações que não toleram atrasos causados pelo acesso a disco.

Os métodos de acesso e algoritmos de processamento de consultas que são eficientes para os DRDBs podem não ter as mesmas vantagens em um IMDB. Na memória principal, a eficiência do espaço é crítica. Assim, métodos de acesso como *B-tree*[3] podem não ter vantagens em um IMDB, porque ele armazena duas vezes mais chaves e ponteiros que outros índices. Da mesma forma, o acesso sequencial suportado pelo *B-tree* perde seus atrativos quando o dado reside em memória. Finalmente, algoritmos de processamento de *queries* que criam grandes resultados intermediários, como *sort-merge join*, não são apropriados quando a residência em memória deve ser preservada. Segundo a autora, o controle de concorrência e recuperação de perdas são os problemas mais difíceis em um IMDB. Isso porque, embora os princípios de serialização e confirmação permaneçam aplicáveis aos IMDB, o gerenciamento de transações requer novos algoritmos, quando a cópia primária do dado reside na memória principal.

[3] *B-Tree* é uma estrutura de dados, pertencente ao grupo das árvores, muito utilizada em bancos de dados e em sistemas de arquivos.

Raatikka (2004), por sua vez, fez estudos de métodos gerais que melhoram a localização dos dados nas estruturas de dados e como esses métodos podem ser aplicados em índices de bancos de dados, visando diminuir o tempo gasto com as ausências de dados ocorridas nos níveis intermediários do *cache* de memória, provocadas pela crescente diferença de velocidade entre o processador e a memória principal que vem ocorrendo com a evolução do *hardware*.

Ailamaki (2000) relata que essa diferença de velocidade ocorre devido ao aumento do tamanho da memória. Nos últimos dez anos, vários dados das aplicações migraram para a memória, e, dessa forma, o gargalo foi transferido das operações de E/S em disco para a comunicação entre o processador e a memória. O autor complementa que, de acordo com a Lei de Moore[4], a velocidade do processador dobra a cada dois anos. Por outro lado, a latência da memória não segue a mesma curva. Em 1980, o tempo para buscar um dado da memória era comparável ao tempo de execução de uma instrução. Devido ao aumento do tamanho da memória e da velocidade do processador, o tempo de acesso da memória em ciclos de processamento são três ordens de grandeza superior à média do número de ciclos que um processador necessita para executar uma instrução. Portanto, a penalidade associada em trazer um dado da memória é equivalente a perder centenas de instruções.

Ainda segundo o autor, as pesquisas na arquitetura de computadores têm usado tradicionalmente programas muito mais simples que SGBDs na avaliação de novos projetos e implementações. Ainda assim, poderia esperar que aplicações de bancos de dados explorassem completamente as inovações da arquitetura, especialmente a partir da saída do gargalo do subsistema de disco. Infelizmente estudos recentes em diversos SGBDRs comerciais mostram que o comportamento do *hardware* nas cargas de trabalho de bancos de dados é subotimizado quando comparado a cargas de sistemas científicos, e o resultado indica que demais análises são requeridas para identificar o real gargalo de desempenho.

Um problema específico nos bancos de dados em memória está ligado à realização do ajuste das consultas. A falta das operações de E/S em disco como fator de custo

[4] Em 1965, o co-fundador da *Intel* Gordon Moore previu que o número de transistores em um pedaço de silício iria dobrar a cada dois anos, dobrando a velocidade do processador.

dominante significa que é muito mais difícil em um IMDB modelar os custos de uma consulta, pois dependem de fatores nebulosos, como o custo de execução de uma rotina na UCP. Portanto, a otimização de consultas em um SGBD tende a fazer uso de modelos de custos simples que contêm constantes "fixas" obtidas por perfis (LISTGARTEN et al, WHANG et al apud MANEGOLD). Um desafio nesta área é modelar a interação entre o estilo de codificação, fatores de *hardware*, como UCP, e arquitetura de memória e os parâmetros de consulta em uma previsão confiável dos custos de execução em memória principal.

Para Bitton (1986), muitas técnicas utilizadas em *benchmarks* de DRDBs são inadequadas aos IMDBs, em razão de certos parâmetros de desempenho que estão relacionados à suposição de residência na memória, tais como: requerimentos de espaço, na forma de memória real ou virtual, e gerenciamento de memória, pelo sistema operacional e pelo SGDB, são parâmetros críticos no desenho de *benchmarks*. Segundo a autora os IMDBs requerem métricas apropriadas para um conjunto de *queries* de testes representativas.

Segundo Raatika (2004), somente uma pequena parte da memória de um computador necessita de acesso instantâneo. Em outras palavras, 0,001% da memória (o *cache*) é suficientemente rápido, enquanto 98,7% são milhões de vezes mais lentos que o desejado. O restante consiste de uma memória principal moderadamente rápida, cerca de 1,3% do tamanho total da memória. No entanto, as expectativas dos usuários frequentemente colidem com essa realidade, conforme figura 1.1. Uma das razões para essa situação é econômica. O custo do *megabyte* varia muito entre diferentes tipos de memórias. Atualmente o disco rígido é o único meio de armazenamento não volátil nos computadores. Dessa forma, ele não pode ser substituído por grandes quantidades de memória principal volátil.

Os diferentes tipos de memória de um computador constituem uma hierarquia de memória de vários níveis, figura 1.2. No topo da hierarquia estão os registradores do processador e memória *cache*, ambos estão localizados no *chip* da UCP. O tamanho dos registradores é de apenas algumas palavras, enquanto o tamanho do *cache* é de 32 a 2048

kilobytes[5]. O segundo nível de memória consiste da memória principal. A memória principal é colocada no barramento de memória da placa-mãe e seu tamanho varia de 28MB a algumas dezenas de *gigabytes*. Finalmente, na parte inferior da hierarquia de memória existe o disco rígido, cujo tamanho é tipicamente centenas de *gigabytes*. A motivação para esse tipo de hierarquia é a necessidade de o usuário possuir memória rápida, com grande capacidade de armazenamento. Tal recurso não pode ser oferecido, mas pode ser simulado de certa forma, como um conjunto de diferentes memórias cooperacionais, usando o princípio da localidade (ou localidade de referência) (LEBEK, CHILIMBI et al apud RAATIKA, 2004). Supondo-se que apenas uma pequena parte dos dados será muito acessada, é suficiente manter esse dado "quente" rapidamente disponível no *cache* e armazenar outros dados na memória ou no disco.

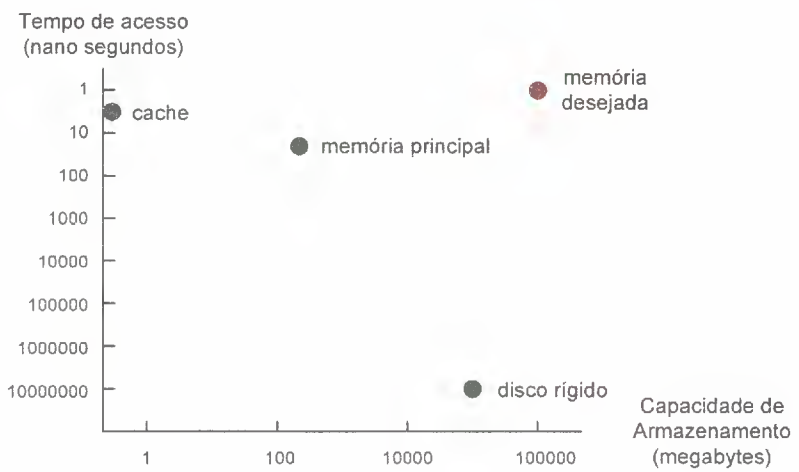

Figura 1.1 – Velocidades e tamanhos dos tipos de memórias (RAATIKKA, 2004)

Cada computador tem um dispositivo de armazenamento não volátil, que geralmente é um disco rígido. Todos os dados são armazenados no disco rígido, de onde as partes solicitadas dos dados são copiadas para o processador, através da memória principal e do *cache*. Se os dados necessários podem ser encontrados, principalmente a partir do *cache* ou da memória principal, o usuário tem a ilusão de ter grandes quantidades de memória moderadamente rápidas sempre disponíveis. A fim de conseguir criar essa impressão, o gerenciamento de memória do computador deve reconhecer os dados mais frequentemente utilizados e mantê-los o mais próximo possível da UCP (ou seja, no mais

[5] A codificação padronizada de um *byte* foi definida em 8 bits, sendo possível representar apenas dois valores em 1 bit (0 ou 1). 1 *kilobyte* corresponde a 1024 *bytes* ou 2^{10} *bytes*.

alto nível possível de memória), usando a hierarquia de memória. Os dados encontrados em um nível da hierarquia são na verdade um subconjunto dos dados encontrados no próximo nível inferior, conforme a figura 1.2.

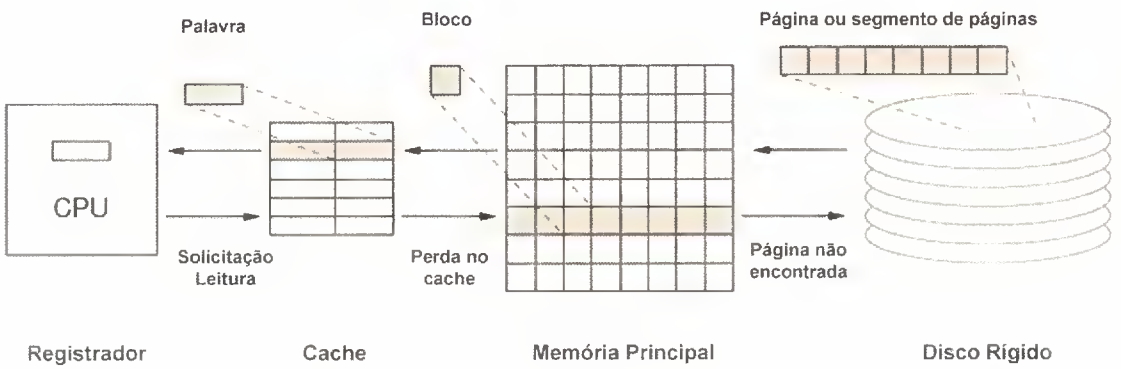

Figura 1.2 – Hierarquia de Memória (RAATIKKA, 2004)

1.5 METODOLOGIA DA PESQUISA

Esse trabalho de pesquisa é por natureza aplicada. A forma de abordagem é quantitativa. Quanto à finalidade ela é exploratória. Está fundada em pesquisa bibliográfica e tem como meio de investigação a pesquisa de campo.

A coleta de dados foi feita com *software Swingbench*, específico para realizações de *benchmarks* em bancos de dados *Oracle*. Como entrada de dados foram informados: o número de usuários concorrentes e o tempo de execução do teste. Além disso, a ferramenta permite definir que os testes sejam iniciados somente após a conexão de todos os usuários e quando as coletas de estatísticas devem iniciar e terminar. Nesse caso, os valores informados objetivaram a duração de cinco minutos de todos os testes, variando apenas a quantidade de usuários. Como saída foram retornados os tempos de conexão, o total de transações concluídas e que falharam, a média de transações por segundo e sua razão máxima, as médias de tempo de UCP no modo usuário[6], modo supervisor[7] e em espera, o total de comandos SQL submetidos às bases e suas médias de tempo de resposta.

[6] Modo Usuário: As instruções privilegiadas não podem ser executadas.
[7] Modo Supervisor: É o modo de execução do processador onde todas as instruções do conjunto de instruções da máquina podem ser executadas.

A análise comparativa dos dados coletados foi feita tendo por base o banco dados *Oracle* (DRDB) e o *Oracle TimesTen* (IMDB), ambos em plataforma *Linux*. Os recursos tecnológicos necessários utilizados para efetuar o experimento e analisar os dados podem ser divididos em:

a) *Hardware*: dois computadores a partir do qual serão feitas as coletas de dados dos sistemas de gerenciamento de bancos de dados avaliados;

b) *Software*: *Linux – software* do sistema operacional, o *Oracle – software* de banco de dados, *Oracle TimesTen – software* de bancos de dados em memória, *Swingbench – software* para testes de bancos de dados *Oracle* e o *Excel – software* para planilhar os dados coletados e gerar gráficos a partir dessas informações

Os resultados do experimento e os dados que aqui constam não são sigilosos e não pertencem a nenhuma organização. As transações simuladas não utilizam dados reais, porém representativos e baseados uma típica aplicação *Online Transaction Processing* (OLTP).

1.6 DESCRIÇÃO DOS CAPÍTULOS

Esse trabalho apresenta um estudo de tecnologia de bancos de dados a ser utilizada em ambientes com intensa necessidade de leitura gravação e encontra-se estruturado da seguinte forma: O capítulo 1 faz uma introdução a alguns conceitos de bancos de dados, sua importância e, principalmente, a necessidade de se ter um bom desempenho para atender às aplicações e aos clientes. Além disso, apresenta um referencial teórico com os trabalhos realizados voltados para o desempenho de bancos de dados em memória e também aborda os objetivos e a metodologia da dissertação. O Capítulo 2 é destinado a detalhar, de forma genérica, a aquitetura dos bancos de dados convencionais e dos bancos de dados em memória, fazendo também uma comparação entre essas tecnologias de bancos de dados, abordando também as propriedades ACID, válidas tanto para os bancos de dados convencionais quanto para os bancos de dados em memória e possui uma sessão final onde são apresentados os aspectos referentes ao desempenho dos bancos de dados, tanto em relação ao IMDB quanto em relação ao DRDB. No Capítulo 3, são detalhados todos os aspectos referentes aos experimentos realizados. O Capítulo 4 apresenta as análises e discussões sobre as resultados obtidos nos experimentos. O Capítulo 5 é destinado à

conclusão da pequisa, incluindo o resumo, objetivos alcançados e os que não puderam ser realizados e o Capítulo 6 apresenta as propostas para pesquisas futuras que não foram abordados nesse trabalho.

2 VISÃO GERAL

2.1 ARQUITETURA DE BANCO DE DADOS

De acordo com Ramakrishnan et al (2000), um sistema gerenciador de banco de dados é um programa elaborado para auxiliar a manutenção e utilização de grandes volumes de dados e a necessidade por esse tipo de sistema, bem como seu uso, cresce rapidamente. O autor apresenta na figura 2.1, uma típica estrutura de um banco de dados baseado no modelo relacional.

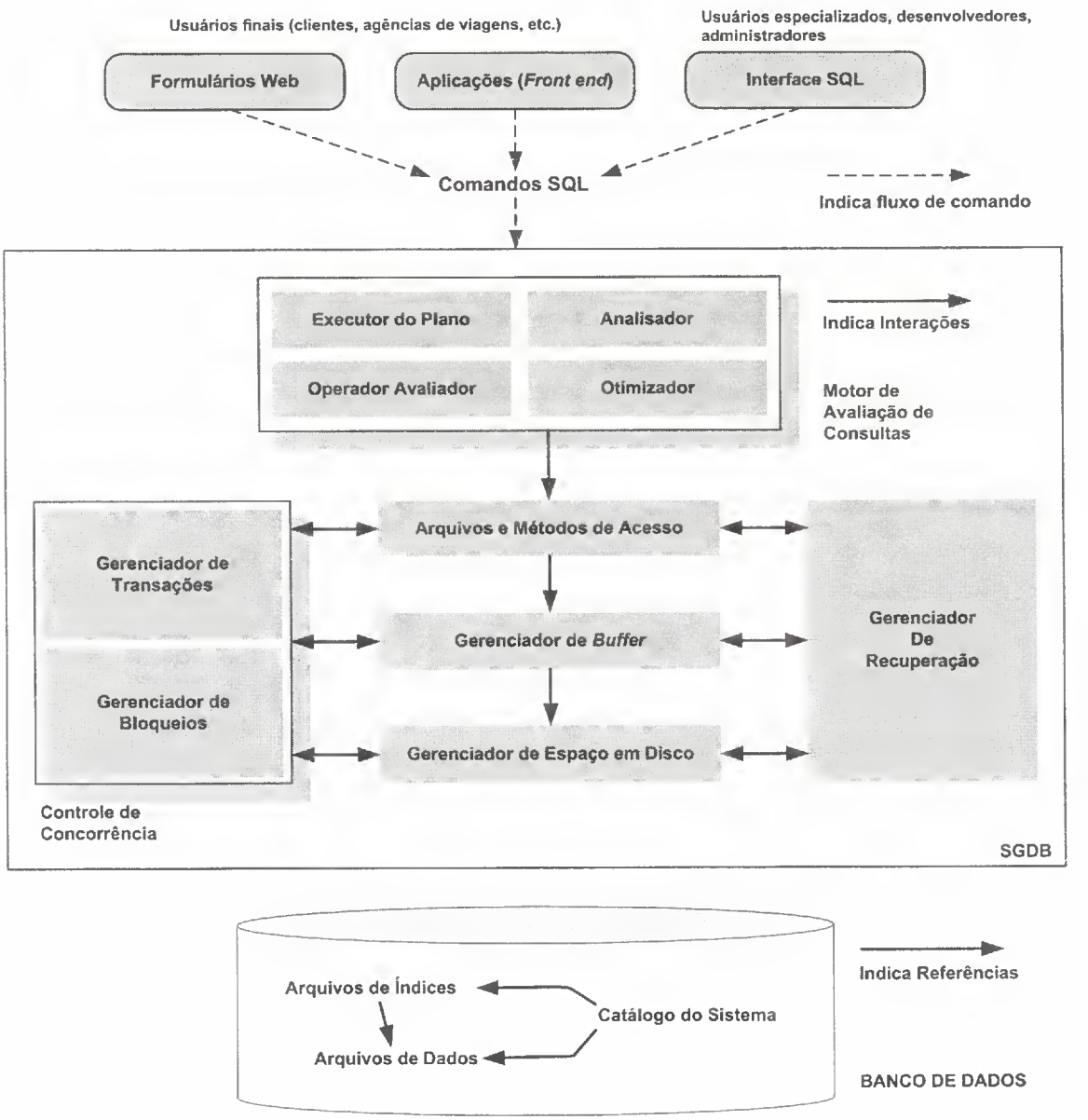

Figura 2.1 – Arquitetura de um SGBD (RAMAKRISHNAN e GEHRKE, 2000)

De forma simplificada, um SGBD geralmente recebe comandos SQL de várias interfaces de usuários, produz planos de avaliações de consulta, executa esses planos no banco de dados e retorna os dados ao solicitante.

Quando um usuário submete uma consulta ao banco de dados, a consulta é analisada e introduzida ao otimizador de consultas, que utiliza informações sobre a forma de armazenamento dos dados para produzir um plano de execução eficiente da consulta. Um plano de execução é um modelo de avaliação de uma consulta e normalmente é representado como uma árvore de operadores relacionais com anotações que contêm informações detalhadas sobre o método de acesso que será utilizado, entre outras infomações.

O conjunto de regras que utiliza os operadores relacionais situa-se no topo da camada de arquivos e métodos de acesso. Essa camada inclui uma variedade de programas que suportam o conceito de arquivos que, em um SGBD, é uma coleção de páginas ou uma coleção de registros. Essa camada tipicamente sustenta uma pilha de arquivos ou arquivos de páginas desordenadas, bem como índices. Adicionalmente, para manter as trilhas das páginas em um arquivo, essa camada organiza a informação dentro da página.

O conjunto de regras da camada de arquivos e métodos de acesso situa-se no topo do gerenciamento de *buffer*[8], que traz as páginas do disco para a memória principal, quando necessárias em respostas às leituras.

A camada mais baixa de uma SGBD é responsável pelo gerenciamento de espaço no disco, onde os dados são armazenados. As camadas mais altas alocam, desalocam, leem e gravam páginas através de rotinas providas por essa camada, chamada de gerenciador de espaço em disco.

O SGBD suporta concorrência e recuperação de falhas através de um cuidadoso controle de agendamento das solicitações dos usuários e da manutenção de registros de

[8] *Buffer* é uma região de memória temporária utilizada para escrita e leitura de dados.

todas as mudanças ocorridas no banco de dados. Os componentes do SGBD que estão associados ao controle de concorrência e recuperação são:

- Gerenciador de Transações - garantem as solicitações das transações e liberam bloqueios de acordo com um protocolo de bloqueios e agendamentos das execuções das transações;
- Gerenciador de Bloqueios - mantém uma trilha de pedidos para bloquear e conceder bloqueios em objetos do banco de dados quando estiverem disponíveis;
- Gerenciador de Recuperação - responsável pela manutenção do registro das transações, e restauração do sistema para um estado consistente após uma falha.

O gerenciador de espaço em disco, gerenciador de *buffer* e a camada de arquivos e métodos de acesso devem interagir com todos esses componentes.

2.2 ARQUITETURA DO BANCO DE DADOS EM MEMÓRIA

Raja et al (2006) relata que, de forma geral, a arquitetura dos bancos de dados em memória é constituída de uma memória principal implementada através de uma RAM padrão e opcionalmente de uma memória não volátil (estável). A memória principal mantém a cópia principal do banco de dados, enquanto a memória estável mantém os registros das transações e uma cópia de segurança do banco de dados. Um processo de geração de arquivos com registros de transações descarrega as alterações realizadas assincronamente no disco. Alternativamente, a memória não volátil pode ser usada como uma memória *shadow*[9]. Essa memória destina-se a manter as atualizações realizadas por transações ativas e no final do arquivo de registros de transações acrescentar informações sobre essas atualizações.

Nos principais produtos comerciais o IMDB é construído para manter todos os dados em tempo de execução residentes na memória RAM além das estruturas de dados e algoritmos de acesso exploram essa característica para melhoria de desempenho. Esses

[9] Memória *shadow*: descreve uma técnica da ciência de computação na qual cada *byte* potencialmente usado por um programa, durante a sua execução, possui um *byte* ou *bytes* sombra. Esses *bytes* sombra são invisíveis ao programa original e são usados para registrar informações sobre um pedaço do dado original.

produtos utilizam os discos magnéticos apenas com o propósito de persistência e recuperação de falhas (ORACLE, 2009).

Para Oracle (2009), as interfaces suportadas por esses produtos são geralmente padrões compatíveis com qualquer outro padrão de um banco de dados relacional. As aplicações emitem comandos SQL através de interfaces *Java Database Conectivity* (JDBC) ou *Open Database Connectivity* (ODBC). Os comandos para definição de bancos de dados, configurações de replicação e grupos de cachê também são aderentes a sintaxe SQL. Além disso, o SNMP é usado para enviar alertas padrões aos sistemas de gerenciamento.

A estrutura física consiste de uma combinação de:

- Bibliotecas compartilhadas;
- Estruturas de dados residentes em memória;
- Processos do sistema;
- Programas administrativos;
- Pontos de verificação (*chekpoint*) e arquivos de registros de transações em disco.

As rotinas que efetuam as operações SQL e funções relacionadas estão embutidas em um conjunto de bibliotecas compartilhadas que os desenvolvedores ligam a suas aplicações e as executam como parte do aplicativo. Essa abordagem difere dos SGBDRs convencionais, que utilizam uma série de programas executáveis nos quais a aplicação se conecta, geralmente em uma arquitetura cliente e servidor.

Normalmente, esses produtos possuem mecanismos de proteção contra potenciais términos anormais da aplicação que possam levar a vulnerabilidade do banco de dados em memória pelo fato das bibliotecas de gerenciamento de dados estarem incorporadas a aplicação, garantindo, dessa forma, que outras aplicações continuem a funcionar sem impacto.

Apesar dos IMDBs utilizarem essa abordagem, as aplicações também podem se conectar ao banco de dados usando o modelo cliente e servidor, embora em vários casos

será obtido melhor desempenho com a ligação direta da aplicação às bibliotecas compartilhadas.

Bancos de dados em memória são mantidos em segmentos de memória compartilhados e contêm todos os dados, índices, catálogos do sistema, *buffers* de registros de transações, bloqueios de tabelas e espaço temporário. Várias aplicações podem compartilhar um banco de dados e uma única aplicação pode acessar vários bancos de dados ao mesmo tempo.

Processos em segundo plano proporcionam serviços de inicialização, término do programa e detecção de falhas da aplicação no nível do sistema. Além de carga, pontos de verificação (*checkpointing*), e controle de paralizações (*deadlock*) no nível do banco de dados.

Programas administrativos são explicitamente acionados pelos usuários, roteiros ou aplicações para executar serviços, tais como: SQL interativos, cópias em lotes, cópias de segurança ou recuperação, migração do banco de dados e monitoração do sistema.

As mudanças nos bancos de dados e os registros das transações são gravadas em disco periodicamente, através de um processo conhecido como pontos de verificação (*checkpoint*), que ocorrem regularmente no tempo estipulado pelo administrador, visando gravar os dados modificados na memória novamente nos arquivos em discos. É possível recuperar o banco de dados através da junção dos pontos de verificações em disco com as transações completadas que ainda estão nos arquivos de registros de transações. Sistemas de arquivos normais em disco são usados para pontos de verificações (*checkpoints*) e arquivos de registros de transações.

A tecnologia mais atraente existente nos IMDBs é a possibilidade de utilizá-lo como *cache* de porções de um SGBDR tradicional residente em disco. Um banco de dados de um grupo de *cache* é criado para manter os dados em *cache*; adicionalmente rotinas de bibliotecas compartilhadas são chamadas pelas aplicações, e um agente do sistema (agente de *cache*) é iniciado para executar todas as transferências de dados entre o grupo de *cache* e o banco de dados tradicional, de forma assíncrona, figura 2.2.

Um grupo de *cache* é uma coleção de uma ou mais tabelas organizadas em uma hierarquia lógica, através dos relacionamentos de chaves estrangeiras e primárias. Cada tabela em um grupo de *cache* está relacionada a uma tabela do banco de dados tradicional. O grupo de *cache* pode conter todos os dados de uma tabela ou apenas um subconjunto de linhas e colunas. Grupos de *cache* suportam as seguintes características:

- Aplicações podem ler e gravar nas tabelas do grupo de *cache*;
- Grupos de *cache* podem ser atualizados (dados vindos do SGBD tradicional) automaticamente ou manualmente;
- Mudanças em tabelas do SGBDR tradicional ou do grupo de *cache* podem ser trilhadas automaticamente.

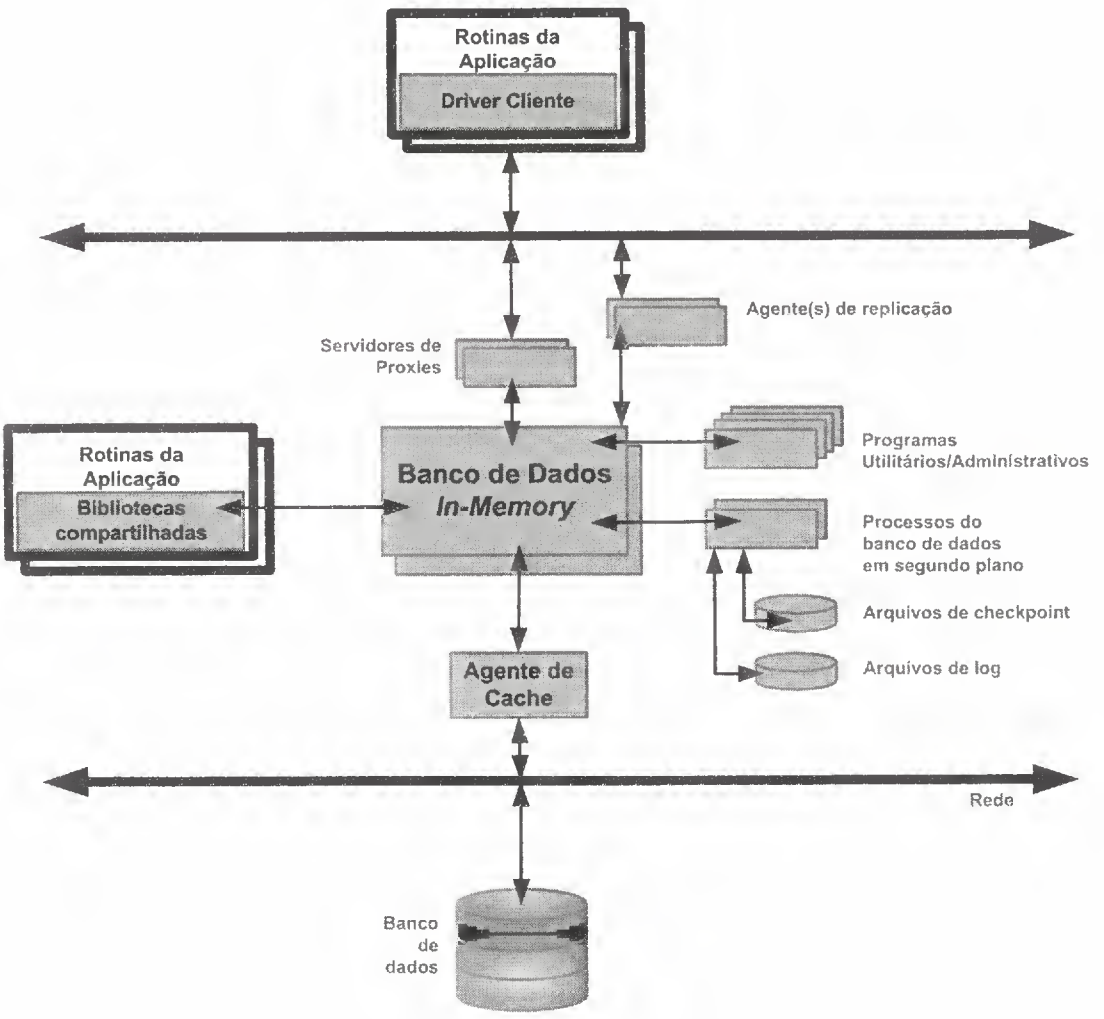

Figura 2.2 – Cache de banco de dados tradicional (ORACLE, 2009)

Quando as linhas de um grupo de *cache* são atualizadas por uma aplicação, suas correspondentes no banco de dados tradicional são atualizadas de forma síncrona, como parte da mesma transação, ou assíncrona, imediatamente após a emissão do comando, dependendo do tipo de grupo de *cache* que foi criado. A configuração assíncrona produz uma vazão significantemente maior e possui tempo de resposta mais rápido para a aplicação.

Se uma transação síncrona por algum motivo falha, toda a transação será desfeita preservando a consistência do banco de dados; enquanto, no caso de uma configuração assíncrona, a falha será percebida somente no banco de dados tradicional e não desfará a transação no IMDB, uma vez que ela já foi efetivada. Entretanto, a vantagem de uma configuração assíncrona é que o IMDB continuará em operação, mesmo que a conexão com o DRDB seja perdida, e as atualizações serão aplicadas automaticamente quando a conexão for restaurada.

As estruturas de árvores de índices *B⁺-Tree*[10], figura 2.3, surgiram como um eficiente meio de pesquisas de valores em registros de bancos de dados tradicionais (RAMAKRISHNAN e GEHRKE, 2000). São estruturas dinâmicas que se ajustam com elegância às atualizações sofridas nos dados; ou seja, permanecem sempre balanceadas. Essas estruturas são na realidade nós (páginas de disco), cujo formato de seu conteúdo, chamado de *entrada*, compreende um dueto {chave, pointeiro} com cada nó podendo armazenar mais de uma *entrada*. Os nós que não são folhas contêm valores de pesquisa e ponteiros e os nós folhas contêm o dado. O principal motivo em utilizar esse tipo de estrutura é a redução de operações de E/S em disco, visto que diversos registros podem ocupar poucas páginas nos nós, seja pela repetição de valores seja pelo tamanho da chave de pesquisa.

[10] *B⁺-Tree*: é uma estrutura de dados, pertencente ao grupo das árvores, considerada uma evolução da estrutura *B-tree*.

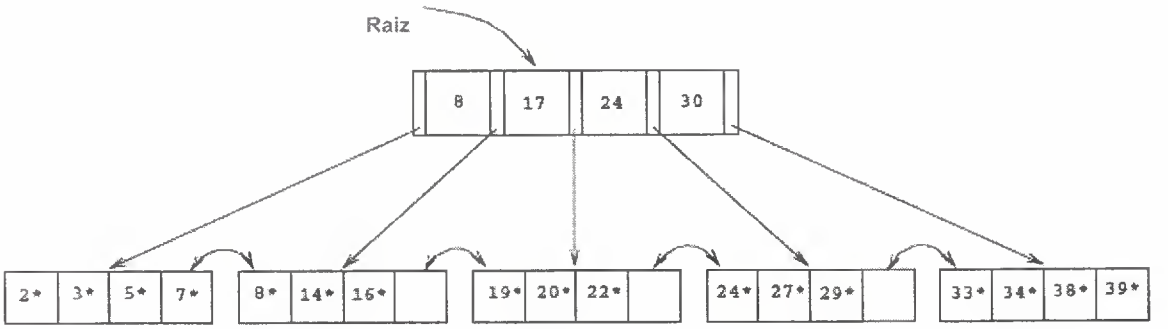

Figura 2.3 - Exemplo de um B^+-*Tree* (RAMAKRISHNAN e GEHRKE, 2000)

Para Oracle (2009), esse tipo de estrutura resulta em uma árvore bem plana e larga. Uma árvore B^+-*tree* realiza isso através das seguintes operações:

1) Mantendo os valores chaves dentro do próprio nó B^+-*tree*, reduzindo, dessa forma, as operações de E/S no disco;
2) Mantendo o maior número possível de entradas de índices nos nós, aumentando o número de entradas B^+-*tree*, que podem ser servidas com uma única operação de E/S.

Essa estrutura, embora ideal quando os dados e índices estão residindo no disco, não é uma boa opção quando os dados residem em memória, uma vez que, quando os dados são mantidos na memória, o objetivo do esquema de indexação deve ser a redução de ciclos de UCP, ao invés do número de operações de E/S em disco.

A *T-tree*[11] é preparada para acesso à memória principal. Ela tem uma *entrada* de índice que é mais econômica que uma B^+-*tree* em termos de tamanho e algoritmo. As conexões entre os nós *T-tree* são obtidas através de ponteiros com valores "menor-que-ou-igual-a" e "maior-que". Esses ponteiros referenciam locais na memória, não páginas no disco. Com apenas duas comparações, o algoritmo de busca *T-tree* sabe se o valor pesquisado está no nó atual ou em outro lugar na memória. E com cada novo nó com ponteiro indireto de índice, a pesquisa é cortada pela metade. É a verdadeira definição de busca binária, figura 2.4.

[11] *T-tree*: é um tipo de estrutura de árvore binária utilizada em bancos de dados em memória.

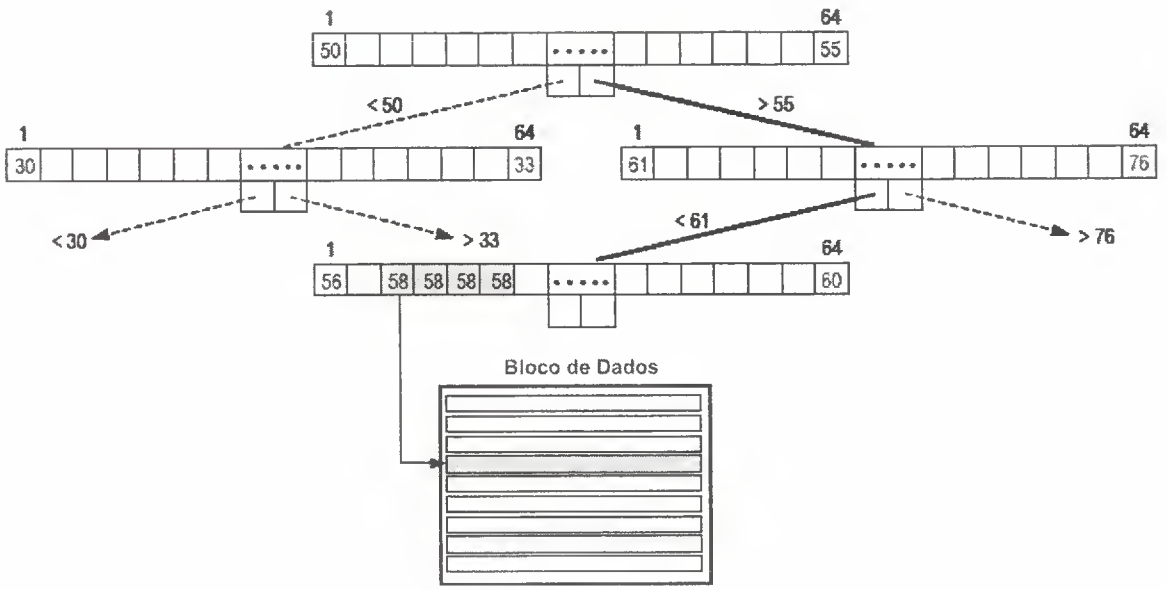

Figura 2.4 – Índice *T-tree* para IMDB (ORACLE, 2005)

A estrutura de índice *T-tree* reduz a necessidade de espaço ao não armazenar o valor chave dentro do nó do índice, simplesmente apontando para o valor que já está residente em memória na linha de dados. Ela não faz operações em disco devido à sua estrutura residir completamente em memória. Ela encolhe a complexidade do código e reduz o caminho no código, sendo um algoritmo consideravelmente simples.

2.3 COMPARAÇÃO ENTRE IMDB E DRDB

Segundo Graves (2002), os bancos de dados em memória eliminam as operações de E/S no disco e existem somente na RAM; mas isso não equivale a simplesmente carregar um banco de dados tradicional em memória principal. Alguns sistemas operacionais possuem a capacidade de criar um sistema de arquivos virtual na memória RAM. No Linux[12], há a solução RAM *Disk,* e no zOS[13] (*mainframe*), a solução VIO[14]. Entretanto, criar um banco de dados nesse tipo de estrutura virtual não proporciona os mesmos benefícios de um IMDB puro. Bancos de dados em memória são menos complexos que os tradicionais SGBDRs, completamente carregados na RAM e utilizam menos CPU e memória.

[12] Termo geralmente utilizado para designar qualquer sistema operacional que utilize o núcleo Linux.
[13] *zOS*: é um sistema operacional de 64 bits para *mainframes*, criado pela IBM.
[14] *Virtual I/O* (VIO): é uma funcionalidade do *zOS* que armazena as páginas de um conjunto de dados na memória.

Comparando um IMDB com um banco de dados tradicional residente em disco, podemos encontrar pelo menos três diferenças chaves:

- Mecanismos de *Caching*;
- Sobrecarga de transferência de dados;
- Processamento de transações.

2.3.1 MECANISMOS DE *CACHING*

Todos os SGBDRs tradicionais incorporam mecanismos de *caching* para manter as mais recentes porções do banco de dados na memória principal, visando reduzir problemas de desempenho introduzidos pela latência de acesso a disco. A utilização da lógica do *caching* inclui:

- Sincronização de *caching*, usada para garantir que uma página do banco de dados na memória principal esteja consistente com sua imagem física em disco;
- *Cache lookup*, determina se o dado solicitado pela aplicação está no *cache*. Se não estiver, a página é recuperada e adicionada ao *cache* para referência futura.

A figura 2.5 demostra como páginas ou blocos de dados são mantidos no *cache*, visando sua rápida utilização pela aplicação em resposta a comandos SQL. No caso, quando uma aplicação submete um SQL para o SGDBR, esse comando é enviado para o processador da consulta que, através de um algoritmo especial, recupera o número do arquivo e da página que contém as linhas solicitadas. Em seguida, para saber se a página já está no *cache*, através de um algoritmo de *hashing*[15], ou seja, de uma função que tem como argumentos o resultado do processador de *query*, é possível gerar um número de uma entrada, geralmente chamado de *bucket#*[16] para, em seguida, varrer as páginas associadas a esse *bucket*, através de uma lista duplamente encadeada.

[15] Hashing: é uma função que gera um número identificador de um intervalo geralmente limitado, ou seja, argumentos de entradas 5 e 98 podem gerar a mesma saída 5.
[16] Associado a cada saída gerada pela função *hash*, existe uma estrutura de *bucket*. Dessa forma, se os valores possíveis de saída da função *hash*, estiverem entre 0 e 99, existirão 100 *buckets*.

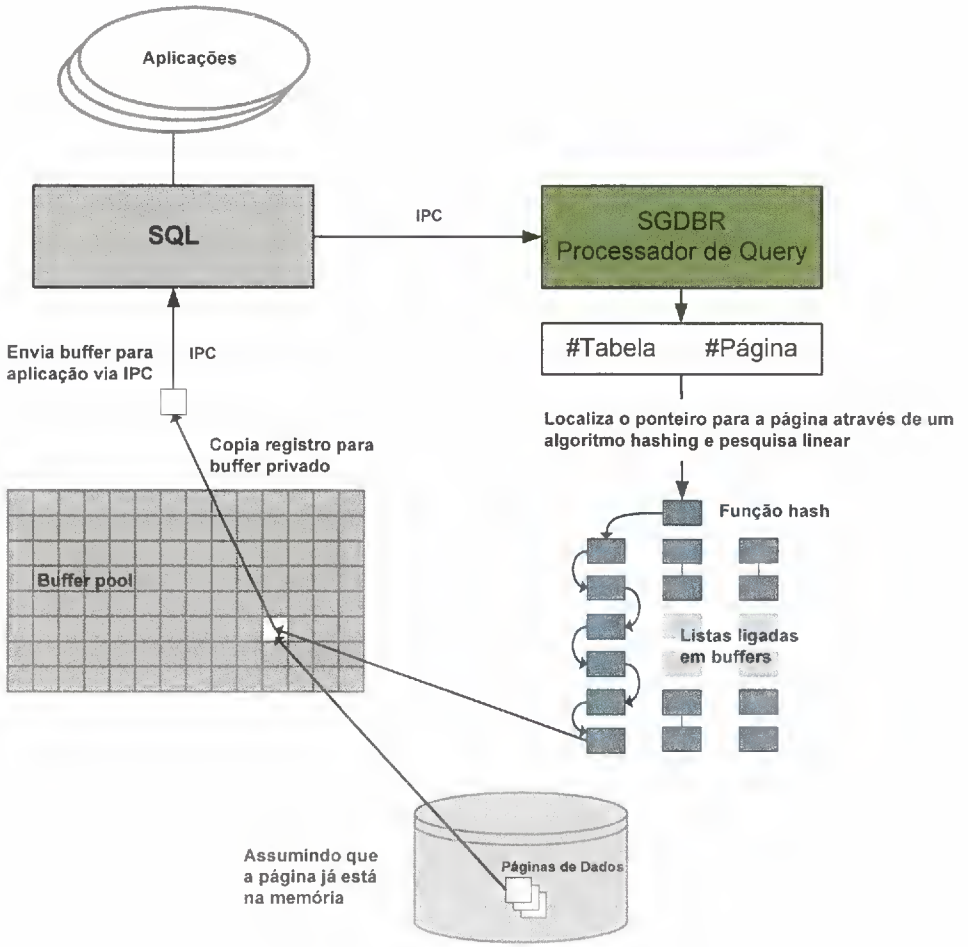

Figura 2.5 – Acesso a registros no DRDB (ORACLE, 2007)

Dessa forma, removendo o *caching*, bancos de dados IMDBs eliminam uma grande fonte de complexidade e sobrecarga de desempenho, reduzindo o trabalho da UCP e, em comparação com um SGBDR tradicional completamente em memória, também a RAM.

A figura 2.6 demonstra como rapidamente um endereço de memória é retornado a uma consulta. Tendo o mesmo ponto de partida, que uma solução tradicional, o comando SQL é enviado ao processador da consulta que, através de um algoritmo especial, recupera o endereço de memória que possui as informações solicitadas. Esse endereço é copiado diretamente na área de memória utilizada pela aplicação; ou seja, a aplicação recebe de forma imediata uma cópia, em sua área de memória, do conteúdo solicitado ao banco de dados.

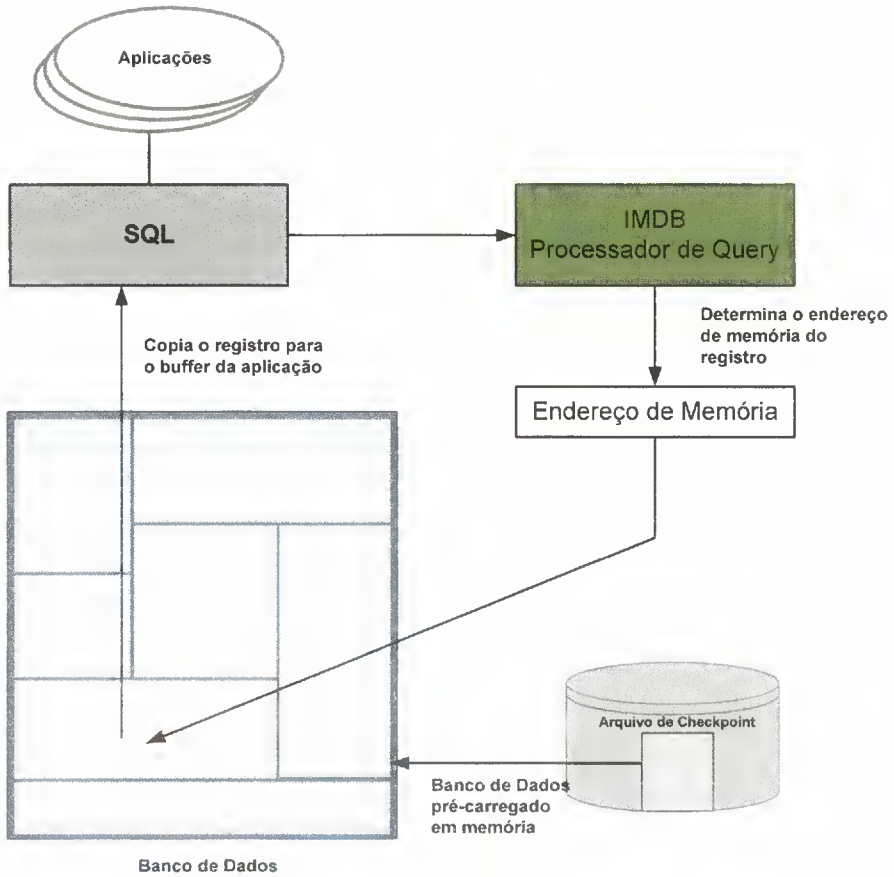

Figura 2.6 – Acesso a registros no IMDB (ORACLE, 2007)

2.3.2 SOBRECARGA DE TRANSFERÊNCIA DE DADOS

SGBDRs tradicionais acrescentam uma notável sobrecarga de transferência de dados devido não só ao *cache* de banco de dados, mas também para o sistema de arquivos e seu *cache*. O processo, ilustrado na figura 2.7, descreve uma aplicação acessando um dado em um banco de dados tradicional residente em disco, modificando seu conteúdo e voltando a gravá-lo na base de dados:

1. A aplicação solicita o dado ao banco de dados através de uma API;
2. O banco de dados instrui ao sistema de arquivos para recuperar o dado da mídia física;
3. O sistema de arquivos faz uma cópia do dado em seu *cache* e envia outra cópia para o banco de dados;
4. O banco de dados mantém uma cópia em seu *cache* e passa outra cópia para a aplicação;

5. A aplicação modifica sua cópia e a envia de volta para o banco de dados através de uma API;
6. O banco de dados copia o dado modificado no seu *cache*;
7. A cópia no *cache* do banco de dados eventualmente será escrita no sistema de arquivos e atualizará seu *cache*;
8. Finalmente o dado é gravado na mídia.

Esses passos não podem ser desligados em um banco de dados tradicional, mesmo quando o processamento ocorre inteiramente na memória. Além disso, esse cenário foi simplificado e não leva em consideração cópias adicionais e transferências requeridas para log de transações.

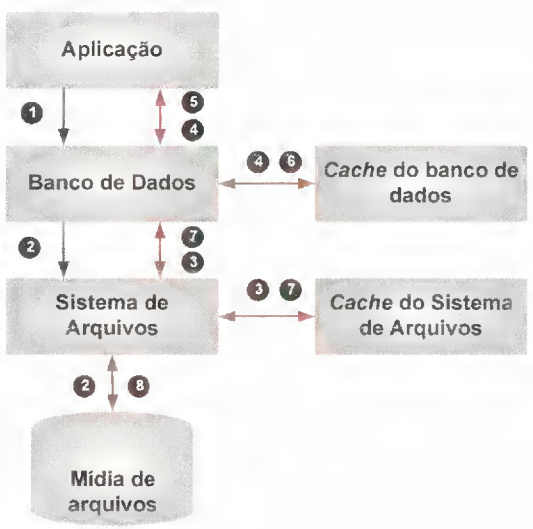

Figura 2.7 – Transferência de dados em Bancos de Dados tradicionais (GRAVES, 2002)

Em contraste com um IMDB que elimina essas etapas e, como consequência, tem pouca ou nenhuma transferência de dados, também não há necessidade de a aplicação copiar dados na memória local, porque o IMDBs fornece à aplicação um ponteiro para o dado que reside no banco de dados. Dessa forma, o dado é acessado diretamente pela aplicação, através de uma API, que garante a segurança de acesso e modificação do dado.

2.3.3 PROCESSAMENTO DE TRANSAÇÃO

Em um banco de dados residente em disco, o processo de recuperação de um desastre ou de uma transação não concluída normalmente é baseado em arquivos de

registros de transações, atualizados toda vez que uma transação é executada. Para proporcionar a integridade transacional, IMDBs mantêm uma imagem atual dos objetos atualizados e, no caso de uma transação ser interrompida, essa imagem é restaurada de forma eficiente. Portanto, outro processo complexo e de uso intensivo de memória é eliminado, quando utilizado um IMDB. A necessidade de adicionar a durabilidade ao sistema é a única razão para manter arquivos de logs de transações.

2.4 PROPRIEDADES ACID DAS TRANSAÇÕES

Segundo Hellerstein et al (2007), o acrônimo ACID – atomicidade, consistência, isolamento e durabilidade – não foi definido formalmente e não são axiomas matemáticos que se combinam para garantir a consistência de uma transação. Ele alerta que é importante distinguir cuidadosamente os termos e seus relacionamentos. Porém, a despeito de sua natureza informal, o acrônimo ACID é útil na organização da discução de um sistema e transações e, dessa forma, é importante saber seu significado.

- Atomicidade – Propriedade que visa garantir que uma transação completa é executada com sucesso ou, no caso de falha de uma única operação de uma transação, nada acontece;
- Consistência – Propriedade cujo objetivo é manter a integridade do banco de dados. Por exemplo: caso uma transação de inclusão viole uma restrição de chave primária, a transação será desfeita;
- Isolamento – Propriedade que garante que a execução de uma transação não afeta outras transações. Em transações concorrentes, onde os mesmos dados são acessados simultaneamente, ocorre um bloqueio apenas dos registros (linhas) envolvidos, sincroniza as transações, a fim de evitar conflitos; as escritas não bloqueiam leituras; as leituras não bloqueiam escritas e controle *deadlocks*;
- Durabilidade – Propriedade para garantir que o resultado do estado da transação deve ser permanente. Transações são registradas em arquivos de logs no disco para, caso seja necessário, sua posterior recuperação.

Enquanto as três primeiras características são comumente suportadas pelos bancos de dados em memória, IMDBs puros, na sua forma mais simples não proveem durabilidade: a memória principal é volátil e todos os dados desaparecem quando a máquina é desligada e ligada em seguida, operação conhecida com "reinicialização".

Com o alto desempenho, IMDBs são boas soluções para aplicações de missão crítica, mas, quando a necessidade de durabilidade das transações se torna necessária, eles podem não ser mais a solução. Para obter durabilidade, sistemas de bancos de dados em memória podem usar diversas soluções:

- Cópias de segurança *on-line*;
- Registro de transações;
- Implementações de soluções de alta disponibilidade
- Memória de Acesso Aleatório não volátil (NVRAM)

2.4.1 CÓPIAS DE SEGURANÇA *ONLINE*

Cópias de segurança *Online* são cópias retiradas da base de dados, enquanto o SGBDR está disponível para leitura e gravação. Essa é a solução mais simples, porém oferece um grau mínimo de durabilidade.

2.4.2 REGISTRO DE TRANSAÇÕES

Um registro de transações é um histórico das ações executadas no sistema de gerenciamento de banco de dados. Visando garantir as propriedades ACID em eventos de quebras ou falhas de *hardware,* o registro deve ser escrito em uma mídia não volátil, normalmente um disco. Se o sistema falha e é reiniciado, a imagem do banco de dados é restaurada de um arquivo com os registros das transações.

O processo de recuperação atua de forma similar ao que ocorre nos bancos de dados tradicionais, desfazendo as transações que ficaram pendentes (*roll-backward*) e aplicando sequencialmente todas as alterações encontradas nos registros de transações (*roll-forward*). Pontos de verificação (*checkpoint*) são usados para acelerar esse processo. Entretanto, essa técnica implica no uso da persistência de memória, como o disco rígido, que é o gargalo, especialmente durante o *resume* do banco de dados.

Embora haja esse aspecto, IMDBs continuam mais rápidos que os bancos de dados tradicionais:

1. Os registros de transações requerem exatamente uma escrita no sistema de arquivos, enquanto bancos de dados residentes em disco não necessitam apenas

escrever no registro, mas também os dados e índices e ainda mais gravações com transações longas.

2. Os registros de transações podem ser configurados com diferentes níveis de durabilidade. Uma troca entre desempenho e durabilidade é permitida através da configuração de mecanismos de registros de transações síncronos ou assíncronos.

2.4.3 IMPLEMENTAÇÃO DE ALTA DISPONIBILIDADE

Alta disponibilidade é um protocolo de arquitetura de sistema e sua associada implementação. Nesse caso é uma replicação de banco de dados primário, com a capacidade de comutar automaticamente (*fail over*) para o ambiente secundário em caso de falha do ambiente primário. Um banco de dados replicado consiste de nós independentes da falha, garantindo que os dados não serão perdidos em caso de falha de um nó. Essa é uma forma efetiva de obter durabilidade de transações em bancos de dados.

De forma similar ao registro de transações, a troca entre o desempenho e a durabilidade é obtida através de configuração de replicação garantida (síncrona) ou não garantida (assíncrona).

2.4.4 MEMÓRIA DE ACESSO ALEATÓRIO NÃO VOLÁTIL (NVRAM)

Para obter durabilidade, um IMDB pode suportar RAM não volátil (NVRAM): Normalmente uma RAM estática garantida com energia ou algum tipo de EEPROM. Dessa forma, o SGBD pode recuperar os dados após uma reinicialização.

Essa é uma opção de durabilidade muito atrativa para os IMDBs. O NVRAM, em contraste com o registro de transações e a replicação de banco de dados, não envolve a latência de operações de E/S em disco e nem a sobrecarga de comunicação.

Independente disso, raramente os fornecedores proporcionam suporte a essa tecnologia. Um dos maiores problemas dessa arquitetura é a limitação de ciclos de gravação nesse tipo de memória, como as memórias *flash*. Por outro lado, existem alguns novos dispositivos de memória que têm sido propostos para endereçar esse problema;

porém, seu custo ainda é proibitivo, em particular se se considerar a ampla necessidade de memória de um IMDB.

2.5 DESEMPENHO DE BANCO DE DADOS

Para Ikematu (2000), Os usuários demandam informações do banco de dados. O SGBD fornece informação para aqueles que a pedem. A taxa entre os pedidos que o SGBD atende e a demanda por informação pode ser denominada desempenho de banco de dados. Cinco fatores influenciam: demanda, *throughput*, recursos, otimização e contenção. O *hardware* e as ferramentas de *software* disponíveis são conhecidos como recursos do sistema. Quando a demanda para um recurso particular é alta, pode acontecer a contenção. Contenção é a condição em que dois ou mais componentes da demanda estão tentando usar o mesmo recurso em modos conflitantes (por exemplo, duas atualizações ao mesmo dado). Se a contenção cresce, o *throughput* diminui. A demanda é o conjunto de transações *online*, processamento em lote, pesquisas *ad hoc*. *Throughput* define a capacidade do computador de processar os dados. Ele é uma composição de velocidade de *E/S*, velocidade da UCP, capacidades de paralelismo da máquina, a eficiência do sistema operacional e o *software* básico envolvido.

Segundo Silberschatz (1999, apud HARTMANN, 2003), a melhor maneira de se ter a média de desempenho é calcular o tempo total para conclusão da carga de trabalho ao invés da produtividade.

Em *Performance Tunning Handbook* (1998, apud HARTMANN 2003), o tempo de resposta por consultas deve estar abaixo de 2% em 80% das vezes. Isto permite a desconsideração de transações mais pesadas. É comum deixar de monitorar por falta de conhecimento, por acomodação ou por falta de tempo, dependendo da quantidade de bancos e atividades proferidas pelos DBA's. O modelo de administração de desempenho criado por Hartmann (2003) possui dois níveis: o gerencial e o técnico. O nível gerencial está dividido em 3 fases: formação de equipe, capacitação e gerenciamento de atividades. O nível técnico divide-se em: definições, implementação e gerenciamento. As definições são os objetivos e papéis a serem desempenhados. Para se chegar a essas definições, seguem-se os passos:

- Instrumentação do sistema: o que medir, onde medir e como medir;

- Monitoração do sistema;
- Caracterização da carga de trabalho;
- Selecionar a alternativa de menor custo com o mais alto desempenho

Wood (2002) afirma que toda organização está sujeita a problemas. Entretanto, nessa nossa "Era do Espetáculo", dois grandes fatores tornam os sistemas ainda mais propícios a desastres: o culto à velocidade nas mudanças e a preocupação cada vez maior com a imagem. "A aceleração e rupturas entraram definitivamente na agenda". A dúvida que persiste é como manter confiabilidade e consistência em ambientes organizacionais em que impera a velocidade. Um segundo ponto soma-se a esses: a preocupação com o sucesso faz com que as pessoas se tornem mais inclinados a realizações rápidas e de impacto (JUNIOR, 2002). O resultado são sistemas mal-elaborados, sem planejamento e com desempenho comprometido.

A pressão do tempo ou do estilo e a meta pela excelência limitam opções e levam-nos a querer saltar etapas. Quando a pressão aumenta, muitos executivos evitam processos mais elaborados e simplesmente copiam soluções de outras empresas, sem considerar a adequação ao seu próprio contexto (JUNIOR, 2002). Isso ocorre, por exemplo, com a compra de muitas aplicações fechadas. Depois de estarem em produção, os problemas de desempenho aparecem e muito pouco se pode fazer, pois não há como mudar o *software*, exceto quando há longos períodos de manutenção deste pelos seus fornecedores.

Nos bancos de dados é comum ocorrer problemas de contenções. Quando esses são encontrados, nem sempre é possível corrigi-los, pois não é possível parar um sistema em produção. Tentar resolver o problema com o sistema "no ar" pode degradar mais o desempenho e ainda causar contenções. As contenções seriam bloqueios de objetos causados pelo acesso simultâneo. Esses bloqueios são também uma maneira de os sistemas manterem a integridade dos dados (WEIKUM, 2000).

Determinar a configuração do *hardware* para uma aplicação é uma tarefa complexa. Os quatro principais recursos a serem dimensionados são (HARRISON, 1997):

- Memória: que é requerida para cada banco de dados e processo de usuário e pelas áreas de memória compartilhada do Oracle.

- Discos: que devem ser suficientes em tamanho para armazenar os dados e suficientes em número para suportar os requerimentos de E/S.
- UCP: que deve suportar os requerimentos do Oracle e dos processos de usuários.
- Rede: que deve suportar a comunicação entre processos em múltiplas máquinas em uma configuração cliente-servidor.

Garantir que há memória suficiente na máquina é essencial para manter um nível de desempenho razoável. A memória pode ser calculada conforme a Equação 2.1 (HARRISON, 1997).

$$Memória = sistema + (\#usuarios * custo\_usuário) + SGA + (\#servidores * tamanho\_servidor) \quad \text{(EQ. 2-1)}$$

Na Equação 2.1 temos os seguintes componentes:

- *Sistema*: é o custo (*overhead*) de memória do sistema. Inclui memória para *kernel*, e *buffers*. Esse valor depende do sistema operacional. Nos servidores Unix pode estar entre 50-150MB.
- *#usuários*: é o número de processos usuários que são residentes no servidor. São comuns em operações em lote.
- *Custo_usuário*: é a memória requerida por cada programa de usuário. Deve-se utilizar um utilitário do sistema operacional para medir isto.
- *SGA*: é o tamanho da Área Global do Sistema. O tamanho adequado da SGA depende da aplicação e ajustar esse valor é uma medida de *ajuste*.
- *#servidores*: é o número de processos servidores do banco de dados implementado no sistema. Para servidores dedicados, esse valor é o número de usuários + o número de processos *background*. Para servidores *multithread*, pode ser 1 servidor para 10 usuários + número de processos de *background*. O número mínimo de processos *background* geralmente é 6.
- *Tamanho_servidor*: é o montante de memória para cada processo Oracle Servidor.

O entendimento sobre o fluxo de processamento das consultas em um banco de dados tradicional residente em disco (figura 2.8), baseado em uma das soluções líderes de

mercado, é fundamental para a compreensão da diferença de desempenho entre essas soluções.

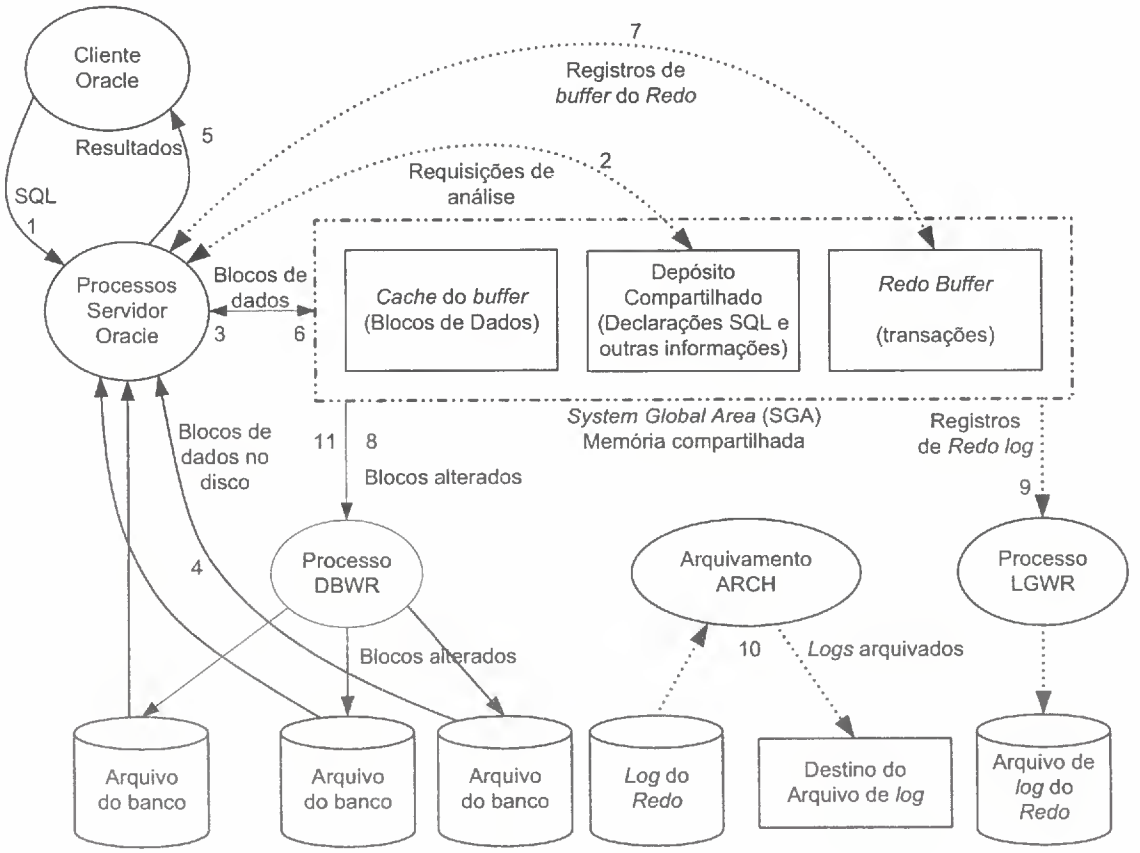

Figura 2.8 – Fluxo dos processos Oracle (HARRISON, 1997)

1. Solicitação de uma consulta (*select*);
2. O processo servidor procura por um comando SQL, no depósito compartilhado, que combine com o recebido. Se não encontrar, o SQL será analisado e armazenado no depósito compartilhado. A análise requer UCP e o armazenamento requer um *latch*;
3. O processo servidor procura no *cache* do *buffer* pelos blocos requeridos. Se encontrar, os blocos de dados devem ser movidos para o final da lista de blocos mais usados recentemente (LRU – *Least Recently Used*). Isso, também requer um bloqueio (*latch*);
4. Se o bloco não for encontrado no *cache* de *buffer*, então o processo servidor irá buscá-lo nos arquivos do disco (*datafile*). Isso requer uma operação de E/S em

disco. Um bloqueio pode ser adquirido antes do novo bloco poder ser movido para o *cache* do *buffer*;

5. O processo retorna as linhas recuperadas para o cliente. Isso pode envolver alguma demora de rede ou comunicação;
6. Se o cliente solicitar uma alteração (*update*), os processos de análise e recuperação das linhas devem ocorrer. Então, ocorrerão as mudanças das informações nos blocos na memória compartilhada e também serão atualizados registros nos segmentos de reconstrução;
7. A alteração (*update*) também fará um registro no *buffer* de *log* do *redo*, que registra os detalhes das transações;
8. Um processo em segundo plano, *database writter* (DBWR) copia os blocos modificados do *cache* de *buffer* para os arquivos de dados. A sessão não tem que esperar que isso ocorra;
9. Quando um *commit* é realizado, um processo em segundo plano *log writter* (LGWR) deve copiar o conteúdo do *buffer* do *log* do *redo* para o arquivo de *log* do *redo*. O *commit* não retornará o controle da sessão até esta escrita estar completa;
10. Se o banco estiver no modo de arquivamento, o processo de segundo plano *archive* (ARCH) irá copiar todos os *logs* de *redo* que ficarem cheios para um arquivo de *redo* arquivado. Um *log* de *redo* não estará legível para uso até ele ser arquivado;
11. Em intervalos regulares, ou quando é forçada a cópia do *log* do *redo* para arquivo, o Oracle faz um *checkpoint*. Isso requer que todos os blocos modificados no cache de *buffer* sejam escritos para disco. Um arquivo de *log* do *redo* não pode ser reusado até o *checkpoint* estar completo;

Claramente é possível perceber que vários algoritmos foram criados para minimizar o acesso ao disco; porém, como as estruturas de memória são "espelhos" das unidades básicas de armazenamento em disco, o processo se torna complexo, elevando riscos de contenções e a consequente perda de desempenho.

A figura 2.9 detalha o fluxo de processamento das consultas em um banco de dados em memória. Entre os componentes que são exibidos na figura, podemos destacar os agentes de replicação, que são utilizados para atualizar outras bases da mesma solução.

Dessa forma, é possível obter alta disponibilidade e equilíbrio no processamento das transações existente na camada de banco de dados. Os programas administrativos que fazem parte da distribuição da solução visam o gerenciamento das bases de dados que venham porventura a serem criadas. E os agentes de *cache* que possibilitam a manutenção das tabelas que fazem parte dos grupos de *cache* criadas no IMDB.

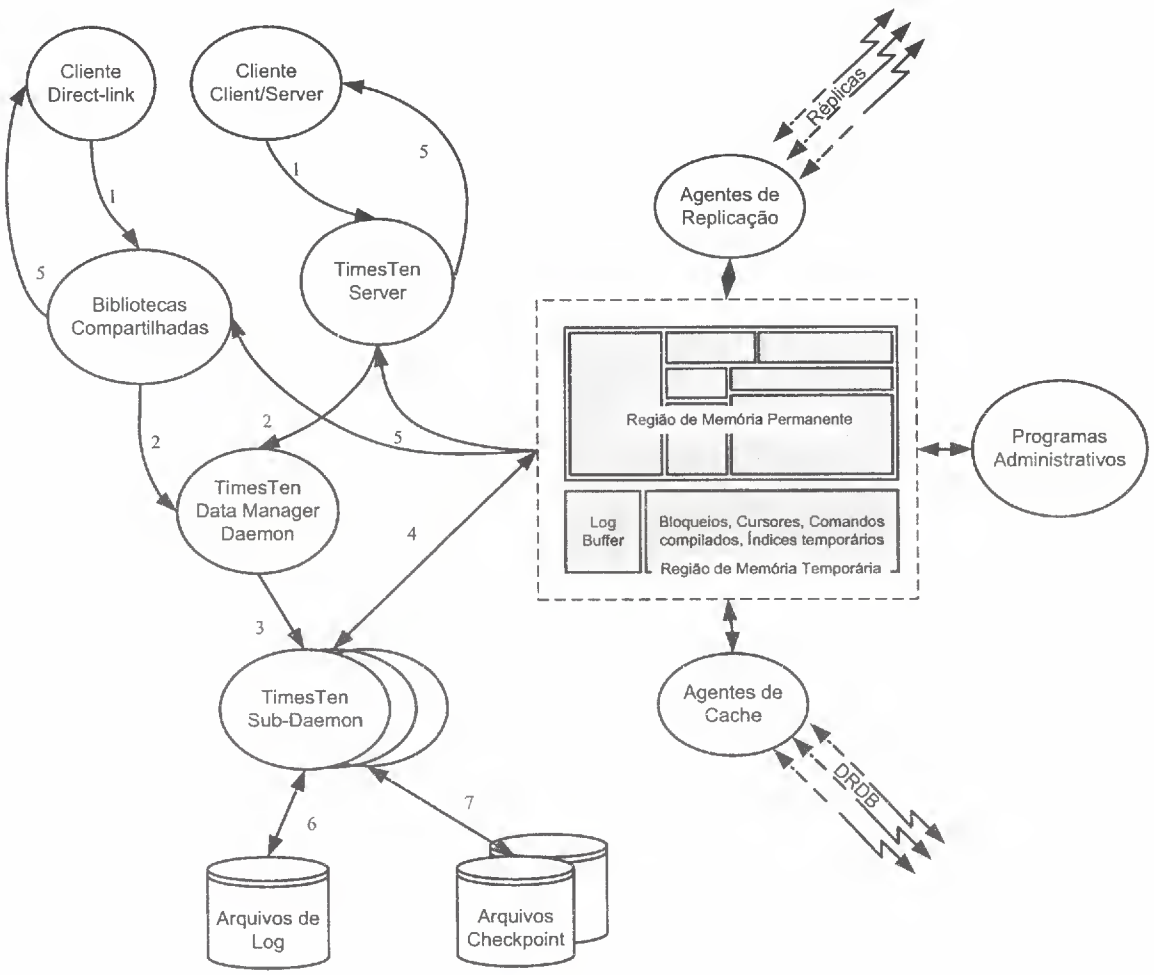

Figura 2.9 – Fluxo dos processos Oracle TimesTen

1. O usuário inicia uma aplicação que fará uma requisição (DML) de acesso ao in-memory database (IMDB). Esssa aplicação pode conectar-se ao IMDB, ligando-se diretamente a bibliotecas compartilhadas ou através de uma arquitetura de rede Client/Server;
2. Para a aplicação acessar o *data store* do IMDB, ela se comunica com o *Data Manager Daemon* de forma transaparente, usando rotinas internas do IMDB.

Uma vez conectada ao *data store* do IMDB, ela mapeia o segmento da memória compartilhada daquele *data store* no seu *address space*;

3. O *Data Manager Daemon* principal cria vários processos *subdaemons* dinamicamente, à medida que eles se tornam necessários.
4. Todo acesso ao *data store* é gerenciado pelos subdaemons. Dessa forma, os DMLs enviados pela aplicação são procurados na área de *cache* de comandos compilados. Se não for encontrado, o *SQL Engine* analisa o comando e armazena nesse cache;
5. Os endereços de memória dos dados requisitados pela *DML* são enviados para a aplicação e suas linhas recuperadas.
6. Há transações que necessitam ser duráveis no *data store*. Nesse caso, o IMDB persiste as transações a cada *commit* ou apenas quando seu *buffer* de transações fica cheio, dependendo da configuração do parâmetro "tipo de atributo de conexão";
7. Em intervalos regulares de 600 segundos, ou sempre que 64 *megabytes* de *log* forem gravados em disco (o primeiro que ocorrer), é feito um *checkpoint*. Esse comportamento pode ser alterado através da configuração de atributos do *data store*;

Nesse ponto, observa-se a simplicidade de construção do acesso aos dados na memória, mantendo-se todas as características de um banco de dados tradicional como acesso de múltiplos usuários, controle de transações, etc.

3 PARTE EXPERIMENTAL

Para se obter o desempenho necessário nos sistemas que utilizam bancos de dados, é preciso antes fazer um planejamento de capacidade dos requisitos da aplicação. Nesse planejamento devem ser respeitados horários de picos e o horário de indisponibilidade. Existem apenas duas possibilidades de um sistema se tornar indisponível: aquela que é permitida ou autorizada e aquela que é fruto de um incidente e normalmente não é bem vista, como, por exemplo, a falta de espaço em disco para o arquivamento dos registros de transações. Nesse caso, todas as transações de inclusão, alteração ou exclusão não são realizadas, e o sistema ficará indisponível, até que seja liberado espaço em disco. O impacto disso às vezes é grande, pois, dependendo da criticidade do sistema, podem ocorrer multas por descumprimento de prazos ou outros tipos de penalidades.

Muitas vezes uma simples cópia de segurança pode ter impacto negativo no desempenho de um sistema, levando-se em consideração a disponibilidade requerida. Assim, é importante conhecer o tempo de indisponibilidade dos sistemas e saber que também existem outros fatores que devem ser analisados.

A busca pelo melhor desempenho de um SGBD requer dedicação, tempo e pesquisa, mas, sobretudo, o conhecimento de todos os fatores que podem influenciar os resultados desejados. Esse é um dos motivos que demonstram como a tarefa de avaliação de desempenho é complexa e deve ser contínua.

Ter uma tecnologia voltada para o desempenho como aliada pode ser muito produtivo e rentável para as organizações, além de facilitar a implantação de soluções de forma rápida e segura. Por isso essa dissertação propõe a utilização de uma nova tecnologia de banco de dados visando obter o desempenho esperado pelas aplicações.

O objetivo, a princípio, não é utilizar essa tecnologia como solução para todos os problemas, mas, sobretudo, como uma alternativa para os sistemas que exigem soluções mais robustas e que não requeiram grandes alterações em seus programas fontes.

Conforme mencionado anteriormente, esse trabalho pretende comparar o desempenho de uma solução de banco de dados tradicional, comercial e de grande reputação com uma solução de banco de dados em memória também comercial e de renome. Para realizar essa comparação de desempenho, os ambientes utilizados foram os mesmos em todos os testes executados em cada um dos SGBDRs, bem como foram utilizados os mesmos programas e arquivos de entrada, e o tempo de medição foi coletado a partir do sistema operacional.

3.1 RESULTADOS ESPERADOS

Espera-se que a tecnologia de banco de dados em memória seja útil na obtenção de níveis de desempenho altamente satisfatórios em aplicações que sofrem com o grande volume de operações de E/S que realizam nos discos.

3.2 DESCRIÇÃO DO EXPERIMENTO

A figura 3.1 descreve o ambiente utilizado nos experimentos realizados. São dois servidores *Dell PowerEdge 6950* com processadores *AMD* com arquitetura *Complex Instruction Set Computer* (CISC) ligados às redes IP, estando um deles também conectado a uma *storage area network* (SAN) para acesso ao sub-sistema de discos externos *CLARiiON CX3* da *EMC*. Os servidores possuem a mesma configuração, sendo um equipamento dedicado aos testes do *Oracle TimesTen* e o outro utilizado para os testes do banco de dados *Oracle* tradicional. Todos os *softwares* são instalados nos discos internos dos servidores. Somente os arquivos que compõem o banco de dados *Oracle* tradicional utilizam os discos externos. O sistema de gerenciamento de arquivos, usado para armazenar os arquivos do banco de dados tradicional é o *Automatic Storage Manager* (ASM), também da *Oracle*. Para usar esse sistema de gerenciamento de arquivos, é necessário instalar o *Oracle Grid Infrastructre*, que configura uma versão *single-node*[17] do *Oracle Cluster Syncronization Services* (CSS), serviço requerido para permitir a sincronização entre as instância do ASM e a do banco de dados Oracle tradicional que depende desse sistema de gerenciamento de arquivos. Três *Logical Units Numbers*

[17] Termo usado para indicar a configuração de um *cluster*, conjunto de computadores que compartilham recursos, de um único servidor.

(LUNs[18]) de 32 GB foram disponibilizados para criação de *disk groups*[19] no ASM. Todos os *disk groups* criados, possuem redundância configurada com proteção externa, ou seja, são garantidos pelo subsistema de discos da *EMC*. As LUNs foram configuradas de acordo com o seguinte critério de distribuição:

- 1 LUN para o *disk group* de dados;
- 1 LUN para o *disk group* de Índices;
- 1 LUN para o *disk group* de *Backup* (*Recovery Area*).

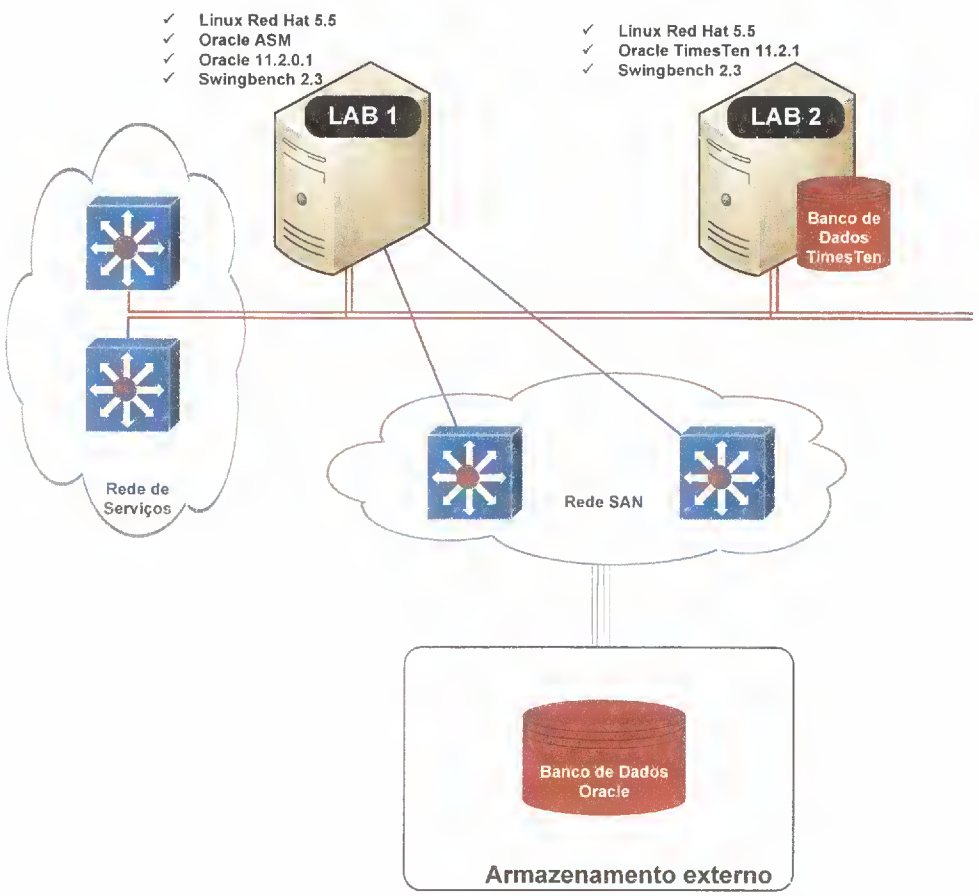

Figura 3.1 – Topologia do ambiente de laboratório

[18] *Logical Unit Number*: é o identificador de um dispositivo endereçado pelo protocolo *SCSI* ou similares como *Fibre Channel* e *iSCSI*.

[19] *Disk Group*: é uma coleção de discos administrados pelo ASM como uma unidade. O conteúdo dos arquivos armazenados em um disk group é geralmente fatiado nos discos para evitar gargalos de acesso aos dados.

A tabela 3.1 relaciona as principais características de configuração dos equipamentos envolvidos nos testes.

Tabela 3.1 – Relação de equipamentos dos testes

	LAB 1	LAB 2
Sistema Operacional	· Red Hat Enterprise Linux 5.5 - 64 bits	· Red Hat Enterprise Linux 5.5 - 64 bits
Discos Internos	· 2 discos de 3,5" SAS (10 mil rpm) de 146 GB formatados em RAID 1; · 3 discos de 3,5" SAS (10 mil rpm) de 300 GB formatados em RAID 5	· 2 discos de 3,5" SAS (10 mil rpm) de 146 GB formatados em RAID 1; · 3 discos de 3,5" SAS (10 mil rpm) de 300 GB formatados em RAID 5
Controladoras RAID	· Controladora de array integrada SAS 3Gb/s – 256MB de memória *cache* PERC 5/e	· Controladora de array integrada SAS 3Gb/s – 256MB de memória *cache* PERC 5/e
Processador	· 4 sockets AMD Dual-Core, 2.8 GHz	· 4 sockets AMD Dual-Core, 2.8 GHz
Memória	· 32 GB DDR3 SDRAM 553 MHz	· 32 GB DDR3 SDRAM 553 MHz
Discos Externos	· 3 LUNs de 32 GB formatados em RAID 5	· Não se aplica
Placas de Rede	· 2 Interfaces de rede 10/100/1000 UTP Onboard · 2 Placa de rede Intel 1000PT Dual Port PCI-e	· 2 Interfaces de rede 10/100/1000 UTP Onboard · 2 Placa de rede Intel 1000PT Dual Port PCI-e
Host Bus Adapter (HBA)	· Placa Fibre Channel 4Gbps Dual Port QLogic 2462 PCI Express	· Placa Fibre Channel 4Gbps Dual Port QLogic 2462 PCI Express

Nos experimentos foram utilizados um DRDB e um IMDB, ambos comercializados pela Oracle:

- DRDB - *Oracle Enterprise Edition* 11.2.0.1 – 64 bit
- IMDB - *Oracle TimesTen* 11.2.1 – 64 bit;

O banco de dados *Oracle* tradicional foi criado durante o processo de instalação do binário e configurado com uma *System Global Area* (SGA) de 8 GB. A SGA é a área de memória intermediária para *cache* dos objetos, comandos SQL e registros das transações. Seu tamanho é suficiente para manter todas as tabelas utilizadas nos testes, na memória RAM principal, visando obter o melhor desempenho possível. Além disso, em função do total de memória RAM do servidor, o tamanho conservador da SGA desse ambiente,

aproximadamente ¼ do total, visa garantir que não ocorrerá *swap*[20] de páginas de memória para disco.

O acesso aos bancos de dados em memória utilizados nos testes, necessita ser feito através de um *Data Source Name* (DSN). DSN é um nome lógico que identifica um banco de dados *Oracle TimesTen* e um conjunto de atributos que são usados na conexão à base. Nos ambientes dos testes, DSN de usuários (privados para os usuários que os criam) estão definidos no arquivo $HOME/.odbc.ini[21]. Uma vez criada a entrada correspondente ao banco de dados no DSN, a primeira conexão realizada nesse banco criará a base e os arquivos que serão utilizados para *checkpoints* e registros de transações. O DSN foi criado conforme exibido na figura 3.2.

```
[ttlabdb]
Driver=/u01/app/timesten/TimesTen/tt1121/lib/libtten.sl
DataStore=/u01/app/timesten/TimesTen/tt1121/info/labdb/ttlabdb
PermSize=4000
TempSize=2000
PLSQL=1
DatabaseCharacterSet=US7ASCII
```

Figura 3.2 - Configuração do DSN do IMDB

3.3 TESTES REALIZADOS

A carga de trabalho usada nos testes está apoiada no programa *Swingbench,* através do *benchmark Stress Test,* previamente distribuído com o produto. O *Swingbench* é baseado no modelo cliente/servidor com transações comandadas da camada cliente para a camada servidor, através de *driver* JDBC. Nos testes, em cada um dos servidores, ao executar o programa, são fornecidos os seguintes parâmetros:

- O número de usuários concorrentes - valor variado a cada execução;
- Tempo de execução - valor estipulado em 5 minutos para todos os testes;

[20] Uma das funções que compõem a memória virtual dos computadores, que possibilita uma aplicação utilizar mais memória que a fisicamente existente.

[21] *$HOME*: é uma variável de ambiente cujo conteúdo é o diretório raiz do usuário. No caso o arquivo *odbc.ini* está no diretório de instalação do *timesten*.

- Todos os testes são iniciados somente após a conexão de todos os usuários;
- A coleta de estatística feita após o primeiro minuto com seu término programado para um minuto antes do fim da execução do teste.

A figura 3.2 apresenta os parâmetros informados em um benchmark.

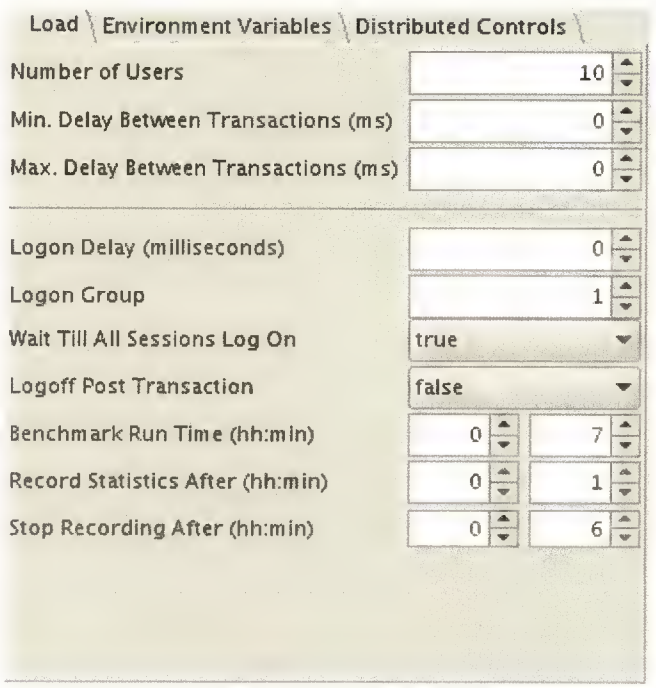

Figura 3.3 – Valores de carga do *benchmark*

O *benchmark,* propriamente dito, executa uma série de transações SQL aleatórias, emitindo comandos de inclusão, alteração, exclusão e leitura em uma tabela. A figura 3.3 apresenta a proporção entre esses comandos nos testes. Os valores configurados representam um típico ambiente transacional, com 65% de escrita e 40% de leitura.

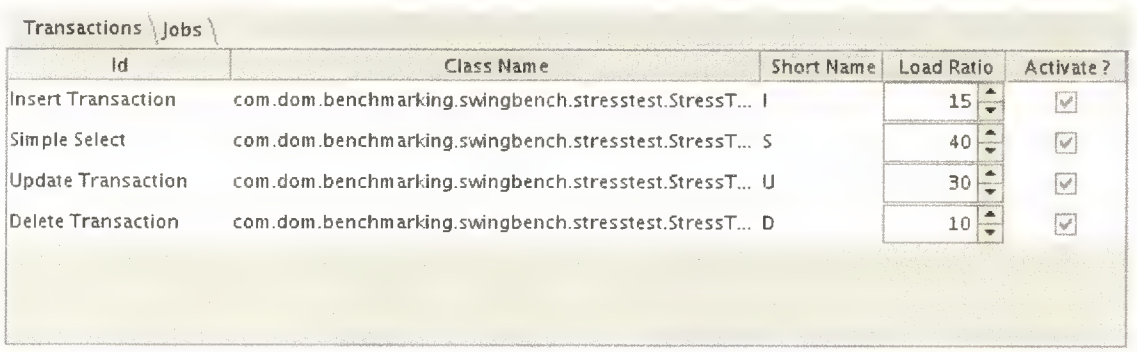

Figura 3.4 – Proporção dos Comandos SQL dos testes

O programa *Swingbench* estabelece a conexão com a base através do esquema SOE, previamente criado nos bancos DRDB e IMDB, com privilégios de conexão e de criação de objetos. Esquemas são entidades lógicas (usuários) que podem criar objetos (tabelas, índices, etc.) no banco de dados, desde que tenham privilégios para isso. A figura 3.4 apresenta os parâmetros de conexão com o banco de dados *Oracle* tradicional. Para a conexão com o banco *TimesTen* é necessário alterar o tipo de *driver* para *TimesTen direct jdbc driver* e a *string* de conexão para o nome do DSN, no caso *ttlabdb*.

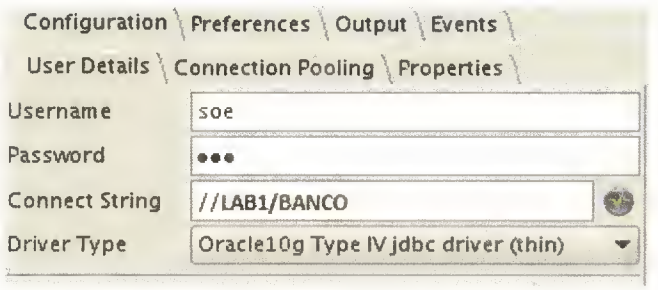

Figura 3.5 – Parâmetros de conexão com o banco de dados

Após a execução dos testes o *Swingbench* gera um relatório no formato *Extensible Markup Language* (XML) com uma série de informações coletadas durante sua execução, tais como, total de transações processadas, tempo de resposta obtido, consumo de UCP para os modos usuários e supervisor, entre outros. Para uma melhor avaliação quanto à diferença de desempenho entre o IMDB e o DBDR, as baterias de testes, com duração de 5 minutos cada, consideram o aumento progressivo da quantidade de usuários simultâneos que acessam o banco de dados. Os valores utilizados foram: 10, 40, 70, 100, 130, 160, 190, 220, 250, 280, 310 e 340. A análise dos resultados encontra-se no capítulo seguinte.

4 ANÁLISE E DISCUSSÃO

Neste capítulo serão apresentados os resultados obtidos nos testes realizados nos respectivos ambientes.

4.1 COMPARATIVO DE TRANSAÇÕES NOS BANCOS DE DADOS

O *benchmark* utilizado no *Swingbench* permite avaliar a carga das transações submetidas aos bancos de dados em memória e convencional, através da coleta de estatísticas. A tabela 4.1 apresenta o total de transações de leitura, inclusão, atualização e exclusão coletadas nos dois ambientes de testes. Pode-se confirmar que a proporção configurada na ferramenta, informada anteriormente, foi seguida em todos os testes.

Tabela 4.1 – Total de Transções por Comandos SQL

	Oracle TimesTen				Oracle Enterprise Edition			
Quantidade de Usuários	Total de Leitura	Total de Inclusões	Total de Atualizações	Total de Exclusões	Total de Leitura	Total de Inclusões	Total de Atualizações	Total de Exclusões
10	8.532.250	3.200.258	6.399.560	2.134.007	487.431	182.086	364.982	120.935
40	6.357.733	2.385.084	4.769.124	1.588.946	765.483	287.115	571.760	191.422
70	5.936.533	2.228.366	4.451.460	1.485.775	736.249	275.527	550.438	183.961
100	5.478.524	2.053.779	4.106.427	1.371.054	700.043	262.723	525.985	175.598
130	5.406.907	2.028.955	4.057.202	1.348.626	727.890	272.364	545.248	182.576
160	5.072.690	1.902.931	3.803.808	1.268.933	625.295	234.522	468.940	155.676
190	5.071.292	1.901.325	3.802.624	1.267.696	693.745	259.890	519.425	173.291
220	4.989.670	1.870.448	3.743.065	1.246.417	681.582	254.996	511.024	170.862
250	4.986.520	1.871.347	3.743.746	1.245.933	585.890	219.469	440.239	146.828
280	4.832.704	1.812.219	3.622.076	1.207.542	581.326	218.046	436.601	145.209
310	4.688.886	1.757.747	3.520.874	1.171.225	568.719	213.924	425.053	142.626
340	4.840.640	1.817.233	3.625.246	1.209.610	513.476	193.463	383.881	128.686

Na tabela 4.1 percebe-se uma tendência de dimunição da quantidade de comandos submetidos aos bancos de dados, tanto o IMDB quanto o DRDB, com o aumento do número de usuários. Esse comportamento é melhor visualizado nas figuras 4.1, para o *TimesTen* e 4.2 para o *Oracle* tradicional. Como a diferença de desempenho entre as tecnologias é significativa, seus gráficos estão exibidos separadamente para facilitar a compreensão. Tendo em vista a existência de processos em segundo plano realizando tarefas de *checkpoint*, registro em arquivos de transações, monitoração das sessões e dos processos de usuários, os comportamentos exibidos nos gráficos podem não ser lineares. Fato esse influenciado também pela utilização diferenciada dos recursos de *hardware*, principalmente UCP, nos ambientes de testes e comprovados na figura 4.2.

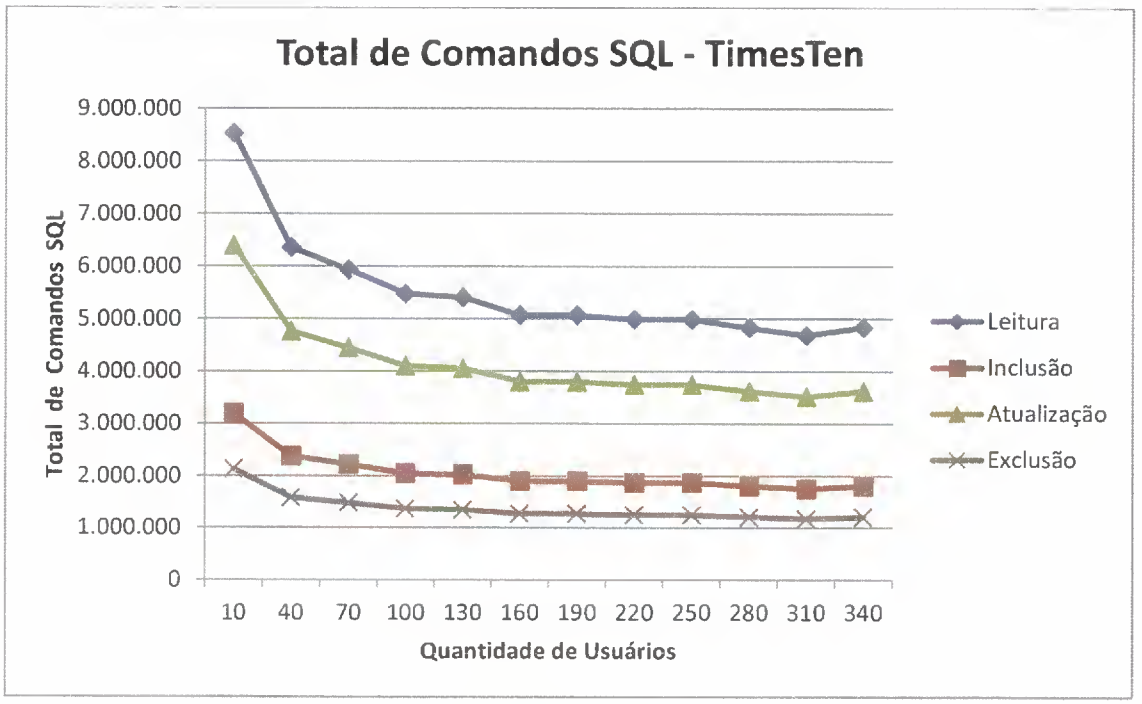

Figura 4.1 – Total de Comandos SQL no TimesTen

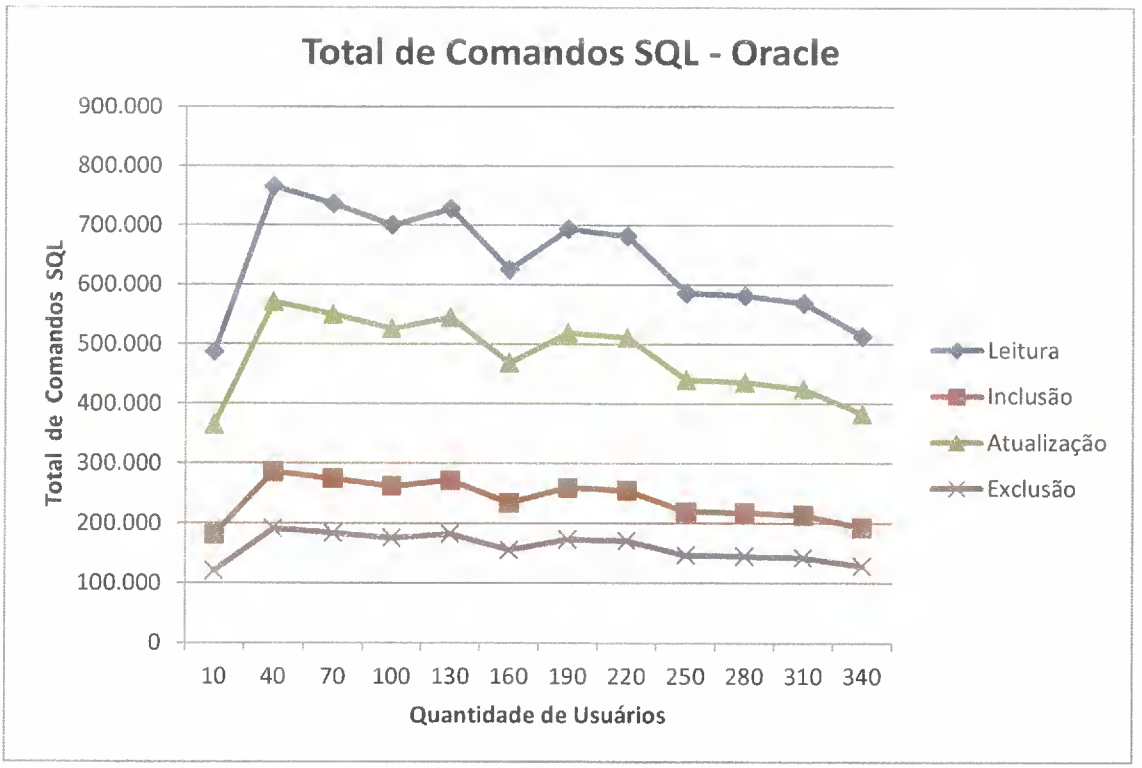

Figura 4.2 – Total de Comandos SQL no Oracle

É importante obsevar que, enquanto o IMDB consegue realizar milhões de transações nos 5 minutos de cada teste, o DRDB realiza centenas de transações no mesmo intervalo de tempo. A tabela 4.2, torna essa informação mais clara, pois apresenta a média de transações por segundo realizadas nos experimentos por usuários.

Tabela 4.2 – Média de Transações por Segundo

Quantidade de Usuários	Média de Transações por segundo TimesTen	Média de Transações por segundo Oracle
10	67.554	3.851
40	50.336	6.053
70	47.007	5.821
100	43.366	5.548
130	42.806	5.760
160	40.161	4.948
190	40.143	5.488
220	39.499	5.395
250	39.492	4.641
280	38.248	4.604
310	37.129	4.501
340	38.309	4.065

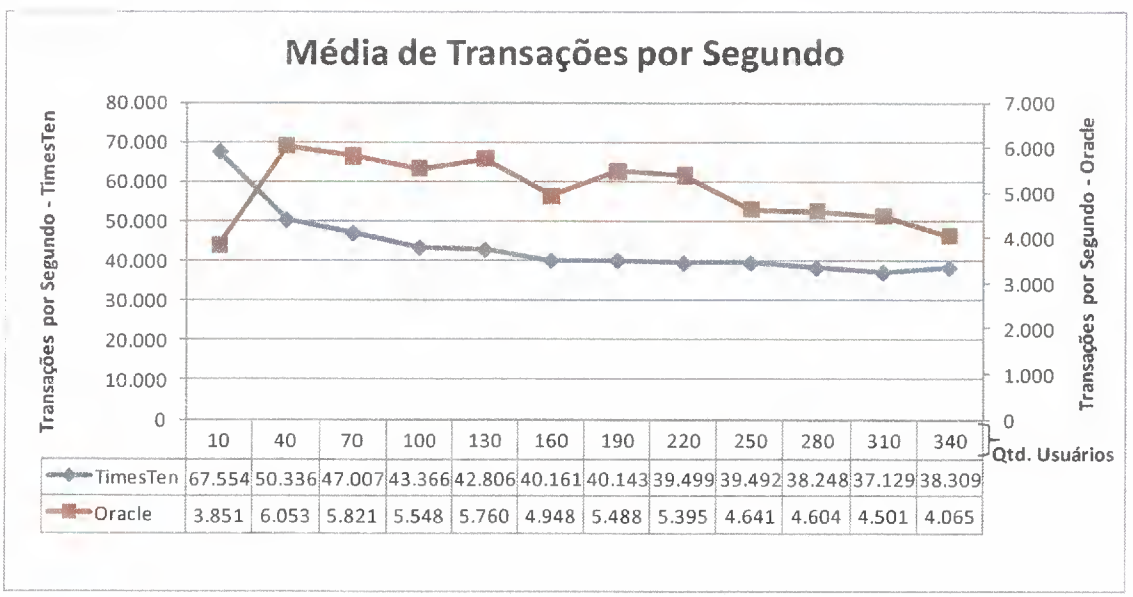

Figura 4.3 – Média de Transações por Segundo

Visando realçar a diferença de desempenho das duas soluções de bancos de dados, a figura 4.3 apresenta esses valores em um único gráfico com dois eixos verticais, um primário para o IMDB e um secundário para o DRDB. A utilização de escalas diferentes visa mostrar que, independente da diferença de transações realizadas em cada uma das soluções de bancos de dados, o comportamento dos dois é similar, com uma tendência de diminuição do número de transações quando a quantidade de usuários simultâneos aumenta.

4.2 COMPARATIVO DO USO DE COMANDOS SQL

A figura 4.4 apresenta o tempo médio de resposta dos comandos de leitura (*select*), em relação ao número de usuários conectados na base de dados IMDB. Observa-se que à medida que o número de usuários aumenta o tempo de resposta também tem uma tendência de elevação.

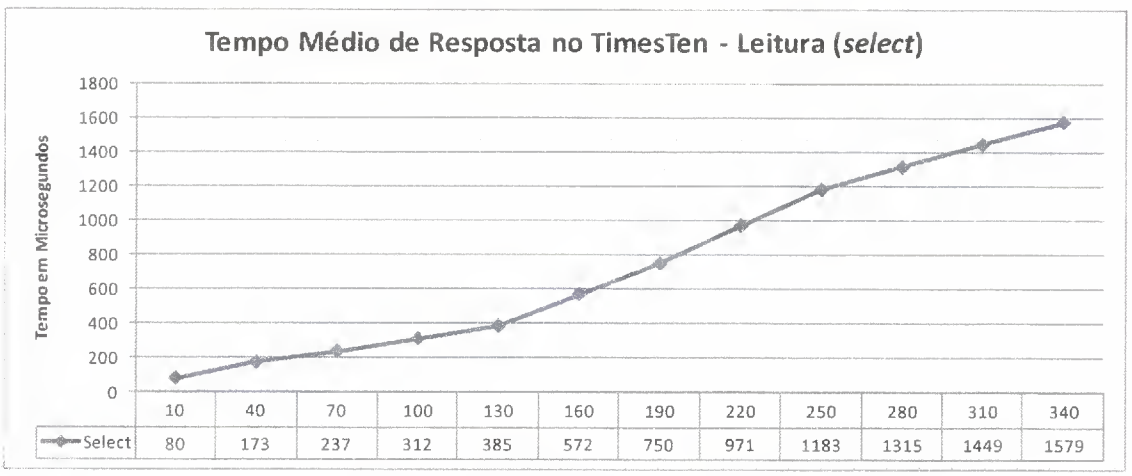

Figura 4.4 – Tempo Médio de Resposta a Leituras no *TimesTen*

Na figura 4.5, é exibido o tempo médio de resposta dos comandos de leitura (*select*) em relação ao número de usuários conectados na base de dados DRDB. Nesse caso, observa-se um comportamento diferente do tempo médio de resposta, não linear, em alguns dos testes, provavelmente em virtude do reaproveitamento de *cache* e a consequente redução do tempo de leitura ou do aumento do consumo de UCP no modo supervisor, provocando uma diminuição do quantitativo de comandos SQL submetidos na iteração.

Independente desse comportamento, o propósito dos testes visa confrontar o desempenho do IMDB em relação ao DRBD. Nesse caso, em qualquer um dos testes observa-se que o tempo médio de resposta das leituras no IMDB é menor que no banco de dados *Oracle* tradicional. No caso da iteração com 10 usuários, por exemplo, temos 80 microsegundos do IMDB contra 771 do DRDB, um desempenho 10 vezes mais rápido no IMDB.

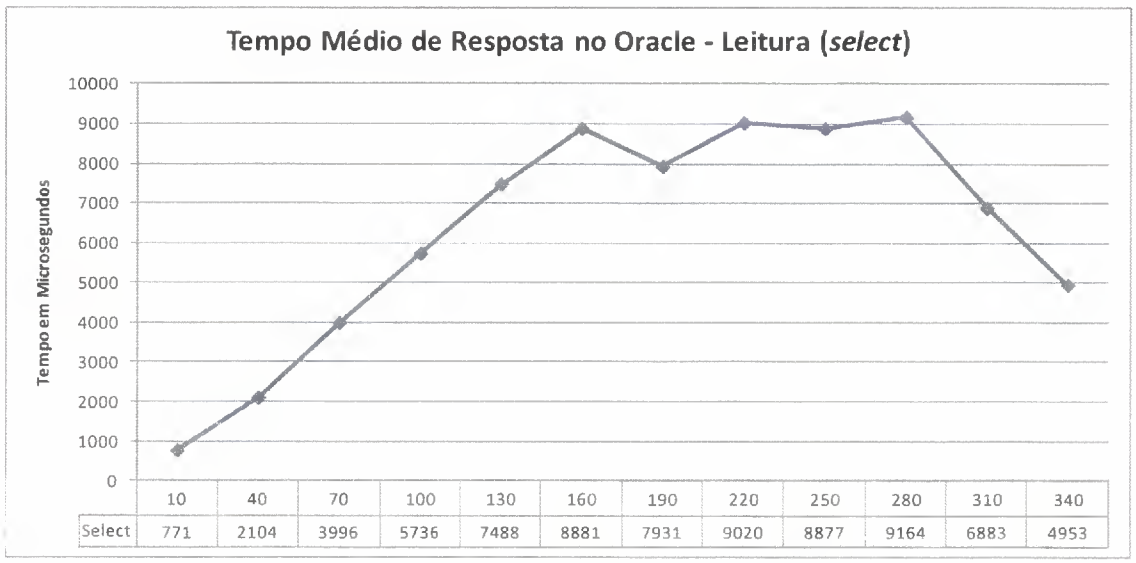

Figura 4.5 – Tempo Médio de Resposta a Leituras no *Oracle*

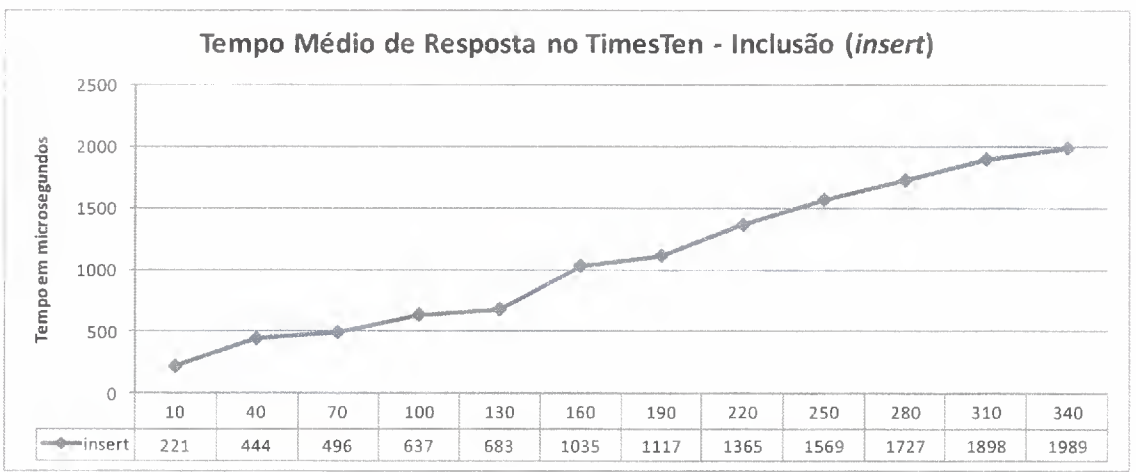

Figura 4.6 – Tempo Médio de Resposta a Inclusões no *TimesTen*

A figura 4.6 apresenta o tempo médio de resposta dos comandos de inclusão de registros na base de dados (*insert*), em relação ao número de usuários conectados no ambiente do IMDB. Observa-se aqui o mesmo comportamento das leituras, ou seja, à medida que o número de usuários aumenta o tempo de resposta também tem uma tendência de elevação.

A figura 4.7 apresenta o tempo médio de resposta dos comandos de inclusão de registros (*insert*), em relação ao número de usuários conectados na base de dados DRDB. Tomando os menores tempos de ambas as soluções para comparação, temos 221 microsegundos no *TimesTen* contra 4724 no *Oracle* tradicional. Outra vez, observa-se uma diferença significativa entre as duas soluções: o IMDB é aproximadamente 20 vezes mais rápido.

Na tabela 4.1, observa-se que na última iteração no banco de dados *Oracle* tradicional, a quantidade de comandos *insert* diminuiu devido ao aumento no consumo de UCP no modo supervisor, demonstrado na próxima seção. Comportamento que justifica a queda no tempo médio de respostas exibido na figura 4.7, na mesma iteração.

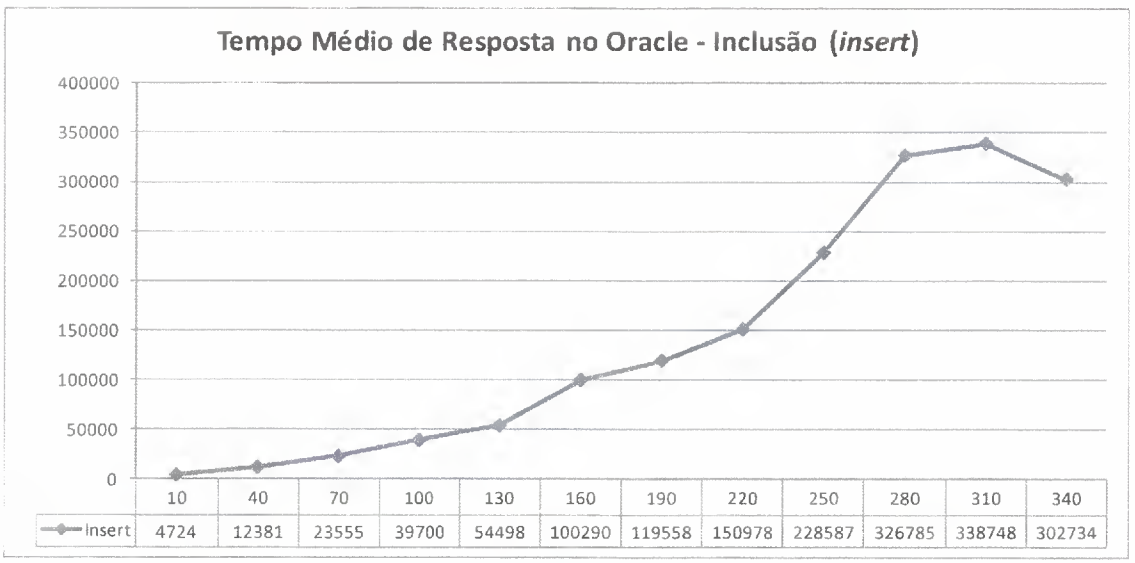

Figura 4.7 – Tempo Médio de Resposta a Inclusões no *Oracle*

A figura 4.8 apresenta o tempo médio de resposta dos comandos de atualização de registros (*update*), em relação ao número de usuários conectados na base de dados IMDB

em cada uma das iterações. Observa-se, aqui, o mesmo comportamento das leituras e das inclusões: o aumento no tempo de resposta está diretamente ligado ao aumento do número de usuários.

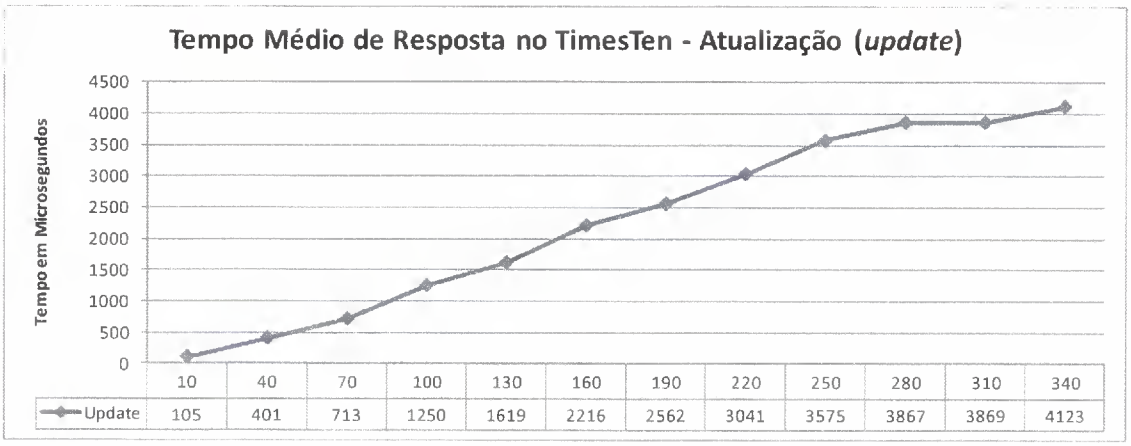

Figura 4.8 – Tempo Médio de Resposta a Atualizações no *TimesTen*

A figura 4.9 apresenta o tempo médio de resposta dos comandos de alteração de dados (*update*), em relação ao número de usuários conectados na base de dados DRDB. Mais uma vez, comparando os melhores desempenhos de cada uma das soluções de bancos de dados, temos 105 microsegundos no *TimesTen* contra 3407 no *Oracle* tradicional, um desempenho aproximadamente 30 vezes superior no IMDB.

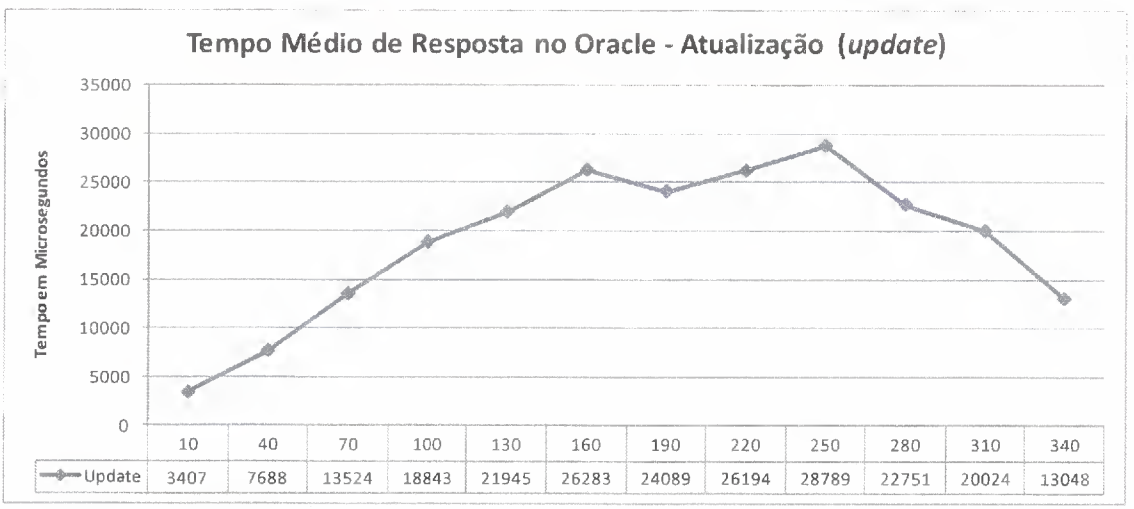

Figura 4.9 – Tempo Médio de Resposta a Atualizações no *Oracle*

Assim como ocorreu nas transações de inclusão, as atualizações no banco de dados *Oracle* tradicional também tiveram iterações com queda do tempo de resposta, apesar do

aumento do número de usuários. Nessas iterações, também observa-se na tabela 4.1 que a quantidade de comandos *update* reduziu devido ao aumento no consumo de UCP no modo supervisor, demonstrado na próxima seção e exibido na figura 4.9.

A figura 4.10 apresenta o tempo médio de resposta dos comandos de exclusão de registros da base de dados (*delete*) IMDB, em relação ao número de usuários conectados. Observa-se aqui o mesmo comportamento das leituras, inclusões e alterações. Eleva-se o número de usuários, consequentemente o tempo médio de resposta.

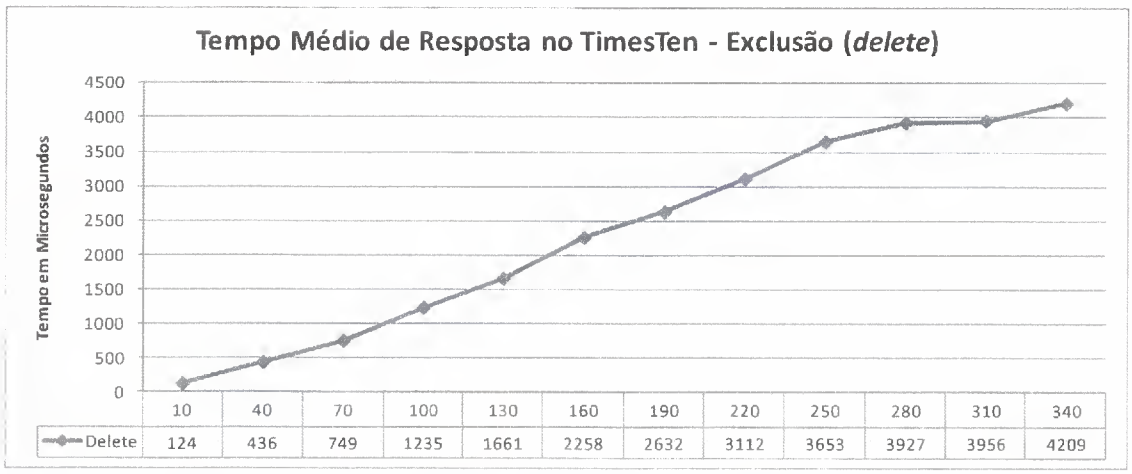

Figura 4.10 – Tempo Médio de Resposta a Exclusões no *TimesTen*

A figura 4.11 apresenta o tempo médio de resposta dos comandos de exclusão de dados (*delete*) da base DRDB, em relação ao número de usuários conectados. Comparando os melhores desempenhos de cada uma das soluções de bancos de dados, temos 124 microsegundos no *TimesTen* contra 3583 no *Oracle* tradicional. Dessa forma, o IMDB tem um tempo médio de resposta aproximadamente 30 vezes superior ao DRDB.

O mesmo comportamento apresentado nas transações de inclusão e atualização, ocorreu nas exclusões no banco de dados *Oracle* tradicional. Alguns testes tiveram queda no tempo de resposta, apesar do aumento do número de usuários. Podem ser observados na tabela 4.1 que a quantidade de comandos *delete* reduziu devido ao aumento no consumo de UCP no modo supervisor, demonstrado na próxima seção e exibido na figura 4.9.

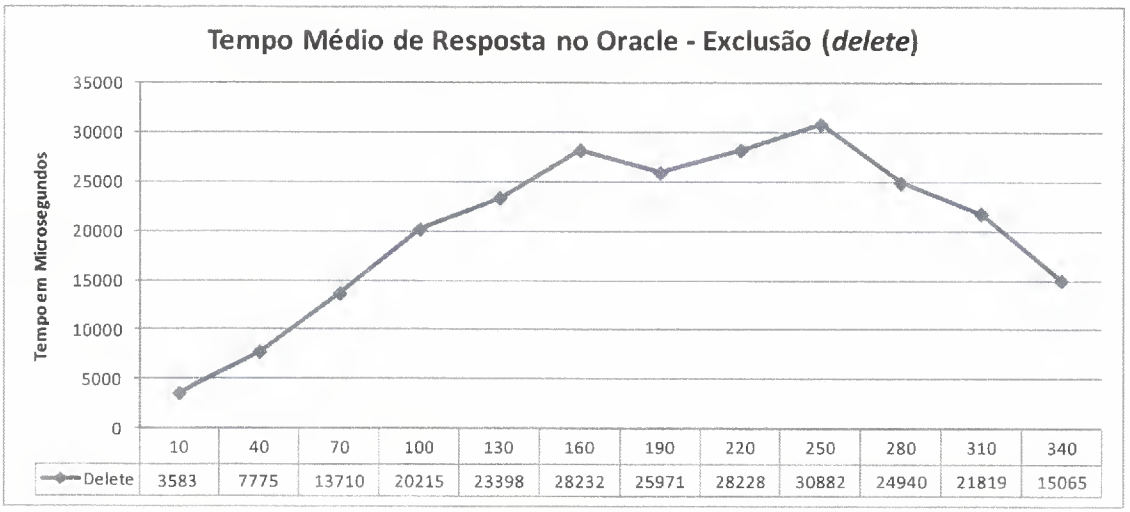

Figura 4.11 – Tempo Médio de Resposta a Exclusões no *Oracle*

4.3 COMPARATIVO DO USO DE UCP

O consumo de UCP esteve constante no ambiente do banco de dados em memória, variando com maior frequência entre os modos usuário e supervisor, no ambiente de banco de dados tradicional.

A figura 4.4 apresenta o comportamento das soluções de bancos de dados em relação ao consumo médio de UCP em modo usuário, supervisor e espera. Observa-se que o uso da UCP pelo *TimesTen* geralmente encontra-se no modo usuário, independente do número de conexões com a base de dados. Isso implica em afirmar que, praticamente, não são executadas instruções privilegiadas que obriguem o chaveamento do processador para o modo supervisor. Outra consequência do comportamento do IMDB quanto ao uso da UCP, é a ausência de espera, também comum em trocas de contexto do processador. Em contrapartida, observa-se o aumento de utilização considerável do processador em modo supervisor e um leve aumento em fila de espera no ambiente do DRDB, provavelmente pela necessidade de execução de instruções no modo supervisor (basicamente E/S).

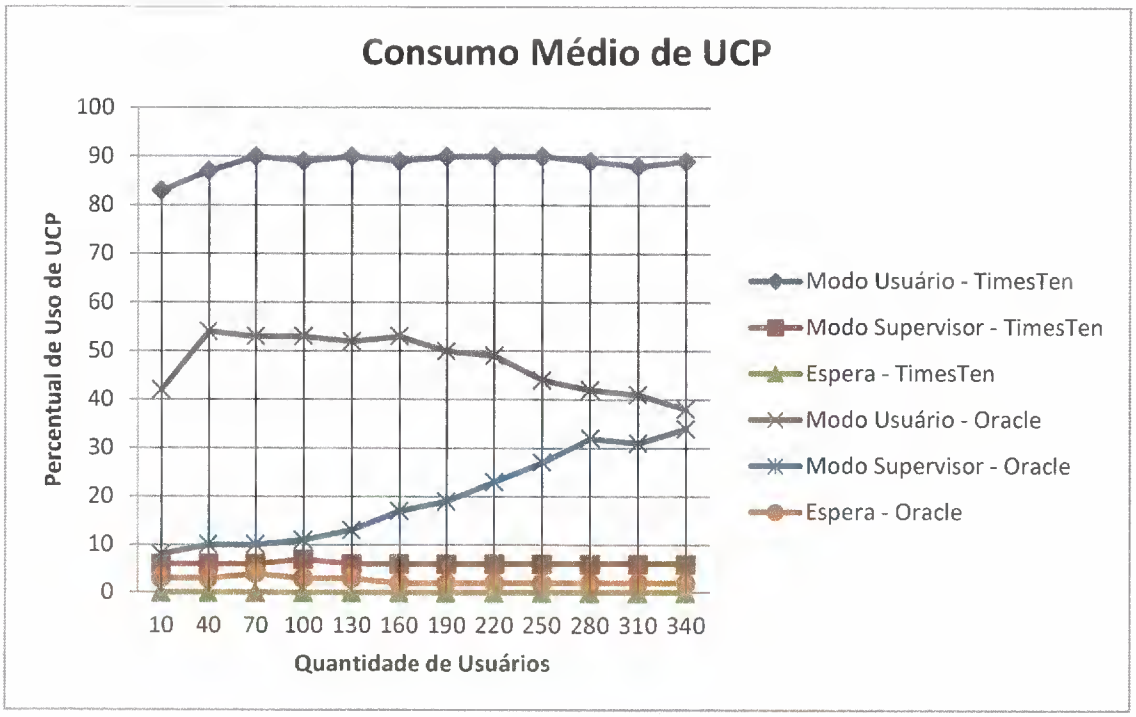

Figura 4.12 – Consumo Médio de UCP

5 CONCLUSÃO

A presente pesquisa foi realizada com o intuito de indicar soluções de bancos de dados em memória, como uma alternativa para aplicações que necessitam de alto desempenho. A empresa na qual o produto foi testado possui um amplo parque tecnológico com crescimento contínuo. Em virtude disso, os problemas de desempenho vivenciados pela equipe de suporte de bancos de dados são constantes e esse tipo de solução pode ser uma excelente alternativa para ambientes com SLAs extremamente agressivos.

O produto *Oracle TimesTen*, aqui apresentado, visa diminuir as deficiências de desempenho existentes em aplicações que fazem vários acessos aos discos. Na empresa onde o estudo foi realizado, do ramo de telecomunicações, os sistemas que fazem uso de CDRs apresentam essas características; ou seja, quase toda a cadeia de receita dessas empresas pode se valer de soluções IMDB, iniciando na coleta, passando pela mediação até o faturamento. Há também a possibilidade de utilizá-la, em casos muito específicos, como aceleradores de aplicações WEB voltados para o atendimento de clientes.

No capítulo um foram citadas as pesquisas já realizadas sobre o tema em questão. O problema de desempenho foi abordado no capítulo dois, deixando claras as diferenças entre as soluções analisadas. Além disso, foram aprofundados os conceitos sobre as soluções, visando deixar claro os aspectos mais importantes que precisam constar nos resultados esperados e na avaliação. No mesmo capítulo foi feita uma comparação entre o IMDB e o DRDB, com objetivo de esclarecer detalhes mencionados na avaliação dos testes.

Antes da utilização da solução de banco de dados em memória, a única alternativa para melhorar o desempenho das aplicações era entender os comandos do aplicativo enviado ao banco de dados e, a partir daí, ajustá-los de tal forma que pudessem ser evitadas as operações de E/S em disco. Após sua utilização, observa-se que somente com a sua utilização, ou seja, sem envolver nenhum tipo de ajuste, o ganho de desempenho foi significativo e qualifica sua utilização quanto à relação custo/benefício for factível.

O excelente desempenho apresentado pelo banco de dados em memória é acompanhado pela diversidade de arquiteturas possíveis, com destaque para sua implantação como *cache* de bases de dados tradicionais. Nessa configuração é possível desenhar aplicações que utilizem grandes dados de bases de dados com desempenho extremamente satisfatórios.

Pode-se observar que realmente existe uma grande diferença de desempenho entre um banco de dados tradicional com seus objetos totalmente carregados na memória *cache* e um banco de dados em memória. Apesar das soluções tradicionais, durante os testes, executarem pouquíssimas operações de E/S em discos, seu desempenho foi inferior ao IMDB.

A maior mudança que soluções como essas provocam estão mais relacionadas à cultura organizacional e aos procedimentos operacionais do que às dificudades técnicas. Geralmente, nas empresas de telefonia, existem barreiras difíceis de serem ultrapassadas, quando um produto desejado não tem grande penetração nesse mercado. Dessa forma, essa solução, mesmo homologada para implantação nessas empresas, tende a ser subutilizada.

A proposta avaliada demonstrou ser eficaz já que cumpre o objetivo proposto; ou seja, diminui significativamente o tempo de resposta das transações que ocorrem em bancos de dados.

5.1 SUGESTÕES PARA TRABALHOS FUTUROS

A solução avaliada nesse trabalho está fundamentada na utilização de um IMDB comercial. Visando diminuição de custos de licenciamento de *software*, poderia ser realizada uma pesquisa comparativa, abrangendo todas as funcionalidades existentes nos principais produtos IMDB comerciais e *open-source.*

A solução aqui proposta possui limitações que foram identificadas. Ela está limitada à memória física do equipamento onde o produto está instalado. Um trabalho que estude soluções IMDB baseadas em *cluster* de servidores ou *grid* de computadores, poderia apresentar soluções para esse problema.

Finalmente, sendo o desempenho relacionado ao melhor uso dos recursos de hardware e, no que diz respeito a esse tipo de solução, principalmente a memória e UCP, julga-se necessário uma pesquisa detalhada para saber a partir de quantos ciclos de UCP ou da quantidade de memória física, que esse tipo de solução deve ser recomendada.

REFERÊNCIAS BIBLIOGRÁFICAS

AILAMAKI, A. Architecture-conscious - database system. [S.l.]: The University of Wisconsin - Madison, 2000. p. 107.

BITTON, D. The effect of large main memory on database systems. **International Conference on Management of Data**, Washington, 1986. 337 - 339.

CODD, E. F. A Relational Model of Data for Large Shared Data Banks. **Information Retrieval**, v. 13, June 1970.

GARCIA-MOLINA, H.; SALEM, K. "Main Memory Database Systems: An Overview". **In: IEEE Transactions on Knowledge and data engineering**, 4, n. 6, dezembro 1992.

GRAVES, S. In-Memory Database Systems. **Linux Jornal**, Houston, v. 101, n. September, 2002. http://www.linuxjournal.com/article/6133.

HARTMANN, V. Administração de desempenho de ambientes de tecnologia da informação em organizações do Distrito Federal, Brasília, 2003.

HELLERSTEIN, J. M.; STONEBRAKER, M.; HAMILTON, J. **Architecture of a Database System**. 2. ed. [S.l.]: Foundations and Trends in Databases, v. 1, 2007.

IKEMATU, R. S. Realizando ajuste na Base de Aplicações. **Bate Byte 97**, maio 2000.

JUNIOR, T. W. **Executivos neuróticos, empresas nervosas**. São Paulo: Negócio Editora, 2002.

KHAN, M. F. et al. Intensive data management in parallel Systems: A Survey. **Distributed and Parallel Databases**, 7, 1999.

MANEGOLD, S. Understanding, Modeling, and Improving Main-Memory Database Performance, Amesterdam, dezembro 2002.

MIYAZAKI, J. Hardware Supported Memory Access for High Performance Main Memory Databases, fevereiro 2005.

ORACLE. Oracle TimesTen In-Memory Database Architectural Overview Release 6.0. **http://www.oracle.com/technology/products/timesten/pdf/doc/arch.pdf**, 2005. B25267-01.

ORACLE. Oracle TimesTen In-Memory Database Introduction Release 7.0. **http://download.oracle.com/otn_hosted_doc/timesten/703/TimesTen-Documentation/sql.pdf**, 2007. B31687-03.

ORACLE. Extreme Performance Using Oracle TimesTen In-Memory Database. **http://www.oracle.com/technology/products/timesten/pdf/wp/wp_timesten_tech.pdf**, 2009.

PISHARATH, J.; CHOUDHARY, A.; KANDEMIR, M. Reducing Energy Consumption of Queries in Memory-Resident Database Systems. **International Conference on Compilers, Architecture and Synthesis for Embedded Systems**, Washington, 2004.

RAATIKKA, V. Cache-Conscious Index Structures for Main-Memory Databases, fevereiro 2004.

RAJA, F. et al. A Comparative Study of Main Memory Databases and Disk-Resident Databases. **Proceedings of World Academy of Science, Engineering and Technology**, 14, August 2006.

RAMAKRISHNAN, R.; GEHRKE, J. Database Management Systems, Universidade de Michigan, June 2000.

SILBERSCHATZ, A.; KORTH, H.; SUDARSHAN, S. **Sistema de Banco de Dados**. São Paulo: Makron Books, 1999.

WEIKUM, G. The web in 2010: "Challenges and opportunities for database research", 2000.

ANEXOS

A – PARAMETRIZAÇÕES DO LINUX RED HAT

Conteúdo do arquivo de configuração de parâmetros do *kernel* (/etc/sysctl.conf) utilizado nos dois servidores:

```
# Kernel sysctl configuration file for Red Hat Linux
#
# For binary values, 0 is disabled, 1 is enabled.  See sysctl(8) and
# sysctl.conf(5) for more details.

# Controls IP packet forwarding
net.ipv4.ip_forward = 0

# Controls source route verification
net.ipv4.conf.default.rp_filter = 1

# Do not accept source routing
net.ipv4.conf.default.accept_source_route = 0

# Controls the System Request debugging functionality of the kernel
kernel.sysrq = 0

# Controls whether core dumps will append the PID to the core filename
# Useful for debugging multi-threaded applications
kernel.core_uses_pid = 1

# Controls the use of TCP syncookies
net.ipv4.tcp_syncookies = 1

# Controls the maximum size of a message, in bytes
kernel.msgmnb = 65536

# Controls the default maxmimum size of a mesage queue
kernel.msgmax = 65536

# Controls the maximum shared segment size, in bytes
kernel.shmmax = 8417423360

# Controls the maximum number of shared memory segments, in pages
kernel.shmall = 2097152

# Extra parameters Oracle 11g Release 2 installation
# Parametros configurados nas máquinas LAB1 e LAB2
kernel.sem = 2048 64000 5010 200
fs.file-max = 6815744
fs.aio-max-nr = 1048576
net.ipv4.ip_local_port_range = 1024 65500
net.core.rmem_default = 4194304
net.core.wmem_default = 4194304
net.core.wmem_max = 4194304
```

```
net.core.rmem_max = 4194304

# Extra parameters for TimesTen
# Apenas na Máquina LAB2
net.ipv4.tcp_rmem=4096 4194304 4194304
net.ipv4.tcp_wmem=98304 4194304 4194304
net.ipv4.tcp_window_scaling=1
```

Sistema de arquivos montados nos discos internos dos servidores:

Filesystem	Size	Used	Avail	Use%	Mounted on
/dev/sda2	12G	5.1G	6.0G	47%	/
/dev/mapper/vgoracle-lvoracle	39G	14G	24G	37%	/u01
/dev/mapper/vglocal-lvtmp	3.0G	71M	2.7G	3%	/tmp
/dev/mapper/vglocal-lvusr	5.9G	2.4G	3.2G	43%	/usr
/dev/mapper/vglocal-lvopt	3.0G	73M	2.7G	3%	/opt
/dev/mapper/vglocal-lvvar	5.9G	223M	5.3G	4%	/var
/dev/sda1	190M	13M	169M	7%	/boot
tmpfs	16G	7.8G	8.1G	50%	/dev/shm

Arquivo de configuração do /etc/*multipath.conf*, visando balanceamento de carga e disponibilidade contra falha de uma das fibras conectadas ao *storage*:

```
## Use user friendly names, instead of using WWIDs as names.
defaults {
        user_friendly_names yes
}
#
defaults {
        udev_dir                /dev
        polling_interval        10
        selector                "round-robin 0"
        path_grouping_policy    multibus
        getuid_callout          "/sbin/scsi_id -g -u -s /block/%n"
        prio_callout            /bin/true
        path_checker            readsector0
        rr_min_io               100
        max_fds                 8192
        rr_weight               priorities
        failback                immediate
        no_path_retry           fail
        user_friendly_names     yes
}
##
blacklist {
      wwid 360019b90d28336000e96c23052c9ed29
      wwid 360019b90d283360013c4f7515ac7aa78
}
multipaths {
```

```
    multipath {
        wwid                3600601603a801d00049f13ea0e51de11
        alias               ora01
    }
    multipath {
        wwid                3600601603a801d000ec331cd0d51de11
        alias               ora02
    }
    multipath {
        wwid                3600601603a801d00c90790cb0c51de11
        alias               ora03
    }
}
```

B – PARAMETRIZAÇÕES DO BANCO DE DADOS ORACLE

Principais parâmetros utilizados para configuração do banco de dados Oracle.

```
audit_file_dest='/u01/app/oracle/admin/dblab/adump'
audit_trail='db'
compatible='11.2.0.0.0'
control_files='+LABDATA/dblab/controlfile/current.260.727247575'
db_block_size=8192
db_create_file_dest='+LABDATA'
db_name='dblab'
diagnostic_dest='/u01/app/oracle'
dispatchers='(PROTOCOL=TCP) (SERVICE=dblabXDB)'
memory_target=13526630400
open_cursors=300
processes=800
remote_login_passwordfile='exclusive'
sessions=800
```