UNIVERSIDADE DO VALE DO RIO DOS SINOS
CENTRO DE CIÊNCIAS EXATAS E TECNOLÓGICAS
PROGRAMA INTERDISCIPLINAR DE PÓS-GRADUAÇÃO
EM COMPUTAÇÃO APLICADA

Leonardo Ribeiro Machado

ATTuneDB - Uma ferramenta de apoio à sintonia de SGBDs baseada na identificação do regime de operação através de modelo probabilístico.

São Leopoldo

2011

Leonardo Ribeiro Machado

ATTuneDB:
Uma ferramenta de apoio à sintonia de SGBDs baseada na identificação do regime de operação através de modelo probabilístico.

Dissertação apresentada como requisito parcial para a obtenção do título de Mestre, pelo programa de pós-graduação em Computação Aplicada da Universidade do Vale do Rio dos Sinos.

Orientador: Prof. Dr. João Carlos Gluz

São Leopoldo

2011

Ficha catalográfica

M149a Machado, Leonardo Ribeiro

ATTuneDB : uma ferramenta de apoio à sintonia de SGBDs baseada na identificação do regime de operação através de modelo probabilístico / por Leonardo Ribeiro Machado. – 2011.

93 f. ; il., : 30cm.

Dissertação (mestrado) — Universidade do Vale do Rio dos Sinos, Programa de Pós-Graduação em Computação Aplicada, 2011.

"Orientação: Prof. Dr. João Carlos Gluz".

1. Sistemas gerenciadores de bancos de dados. 2. Banco de

Catalogação na Fonte:
Bibliotecária Vanessa Borges Nunes - CRB 10/1556

RESUMO

O desempenho de um SGBD é um fator crítico a ser considerado durante a sua utilização. Diversas técnicas são atualmente empregadas na tentativa de aumentar o desempenho de um SGBD. Esta pesquisa integra tecnologias de agentes e de mineração de dados para a criação de modelos probabilísticos (bayesianos) de decisão aptos a auxiliar no processo de melhoria de desempenho de um SGBD. Este modelo é usado, então, como base da ferramenta *ATTuneDB* de sintonia de SGBD. A partir da carga real de operação de um SGBD PostgreSQL, a ferramenta utiliza este modelo para identificar o regime de trabalho do SGBD e encontrar o melhor conjunto de valores para os parâmetros deste SGBD, apoiando o administrador do SGBD na tarefa de otimizar o desempenho deste.

Palavras-Chave: Sistemas Gerenciadores de Banco de Dados, Mineração de Dados, Redes Bayesianas, Agentes.

ABSTRACT

The performance of a DBMS is a critical factor to be considered while using it. Several techniques are currently employed in an attempt to increase the performance of a DBMS. This research integrates agent technologies and data mining for building probabilistic decision models (Bayesian) able to assist the performance improvement process of a DBMS. This model is used to build the *ATTuneDB* DBMS fine-tuning tool. Receiving information about the real workload being submitted to a PostgreSQL DBMS, and using the probabilistic model, the tool is able to identify the type of the workload, and find the best set of value for the parameters of this DBMS, thus, supporting the DBA on the task of optimizing the DBMS performance.

Keywords: Database Management Systems, Data Mining, Bayesian Networks, Agents.

LISTA DE ILUSTRAÇÕES

SUMÁRIO

1 INTRODUÇÃO

Com a massificação dos SGBDs (Sistemas Gerenciadores de Banco de Dados) através de seu frequente uso por entidades que possuem sistemas computadorizados, cresce também a necessidade de se manter o correto e adequado funcionamento dos mesmos (WIESE et al., 2008). Temas como a segurança da informação e o desempenho do SGBD têm sua importância valorizada à medida que as empresas passam a guardar, se não a totalidade, grande parte de seus dados nestes sistemas. Com o crescimento do volume desses dados armazenados, a figura do profissional de computação torna-se importante para que se possa manter esta estrutura, pois cada vez mais se necessita de um correto monitoramento do SGBD com a finalidade de se tratar os problemas que possam eventualmente aparecer (WIESE et al., 2008).

Profissionais qualificados estão cada vez mais escassos, e com a atual competição entre as empresas, os seus dados passam também a se tornar fonte de preocupação em relação à segurança (MACHADO e NASCIMENTO, 2008). Também o grande número de aplicações acessando simultaneamente estes dados torna mais difícil uma correta interpretação dos problemas que ocorrem no SGBD e de possíveis ações para contorná-los (MACHADO e NASCIMENTO, 2008).

A falta de conhecimentos adequados muitas vezes faz com que as entidades se preocupem mais com tecnologias de rede ou de hardware, supondo que com mais recursos computacionais os problemas se resolvem, quando uma correta configuração dos atuais recursos poderia sanar o problema (MACHADO, NASCIMENTO, 2008).

A correta configuração dos parâmetros de operação do SGBD, também denominada de sintonia (*tunning*) do SGBD, pode resolver questões de otimização e problemas de desempenho, oportunizando ganhos substanciais em relação ao ambiente computacional que hoje se conhece (LOZANO, 2010). Além disso, a sintonia do SGBD, quando possível, oferece uma solução de otimização que não requer a compra e instalação de novos recursos computacionais para o SGBD, algo oneroso e desgastante porque usualmente exige a parada da operação do sistema. O processo de sintonização pode ser projetado para operar sem interferir com a utilização do SGBD e utilizando os recursos computacionais já presentes na instalação.

1.1 Contextualização do Problema

Deve-se ressaltar, entretanto, que a sintonização de um SGBD não é uma tarefa simples de ser realizada. Para que a mesma possa ser executada, é necessário conhecimento especializado na administração de bancos de dados. Para uma correta configuração dos parâmetros de um SGBD, é necessário levar em consideração diversos itens que possam interferir no desempenho e segurança desejados para o banco de dados em questão. Estes itens podem ser enquadrados em algumas categorias (LOZANO, 2010). Há itens que correspondem a fatores do ambiente onde o SGBD está inserido. Aqui se enquadram o sistema operacional sobre o qual o SGBD está sendo executado, os recursos de hardware disponíveis, o próprio SGBD que se está utilizando. Há também os itens que correspondem aos parâmetros disponíveis para mudança em um SGBD. Aqui se enquadram os parâmetros valorados, os quais podem ter o seu valor alterado de acordo com o desejo do usuário. Estes itens referem-se à maneira através da qual o SGBD tratará questões de memória, processamento, armazenamento de dados, etc.

Além disto, é bastante importante se ter em mente os objetivos da sintonia do banco de dados. O que se está buscando? Aqui questões como desempenho e segurança geralmente são os norteadores para uma configuração acertada do SGBD.

Todas estas questões devem ser motivos de preocupação constantes de um administrador de banco de dados (DBA na literatura inglesa) no momento de se efetuar a sintonia do SGBD. Esta não é uma tarefa simples, principalmente pelo número de variáveis envolvidas no procedimento.

Uma ferramenta capaz de auxiliar o DBA na tarefa de sintonizar corretamente um SGBD teria sua importância ainda mais salientada à medida em que esta ferramenta possa ser capaz de se basear em eventos ocorridos com o SGBD em questão para tomar decisões acertadas e assim amparar o profissional.

Apesar de este ser um tema importante de pesquisa atualmente (como pode ser observado nos trabalhos relacionados desta pesquisa), a literatura pesquisada não apresentou um estudo aprofundado sobre as tecnologias e modelos aplicados nesta pesquisa para a sintonia de banco de dados.

O principal diferencial da presente pesquisa é a aplicação de tecnologias de agentes e de mineração de dados para a criação de modelos probabilísticos (bayesianos) de decisão aptos a auxiliar no processo de melhoria de desempenho do SGBD.

1.2 Objetivo da Pesquisa

O objetivo principal da presente pesquisa é o desenvolvimento de uma ferramenta capaz de determinar o conjunto de valores para parâmetros de operação que otimize o desempenho de um SGBD. A determinação destes parâmetros será feita levando em consideração o histórico apresentado pelo SGBD no que diz respeito ao conjunto de transações que está sendo submetido ao mesmo. O conjunto de transações que é submetido ao SGBD caracteriza o regime de trabalho sob a qual o mesmo está submetido.

A ferramenta, denominada *ATTuneDB* (*Agent for automatic DB tunning*), é capaz de interpretar um histórico de fatos ocorridos ao SGBD, como o conjunto de transações executadas no mesmo, e utilizar estes dados para abastecer um modelo de decisão probabilístico cuja responsabilidade consiste em identificar, baseado no histórico utilizado como entrada, o conjunto ideal de valores dos parâmetros que devem ser utilizados no SGBD em questão, de forma que o desempenho obtido seja aumentado. A ferramenta sugere as alterações necessárias para permitir que o SGBD funcione de maneira otimizada no que tange o aproveitamento dos recursos computacionais de hardware disponíveis no ambiente onde o mesmo está inserido.

Para atingir tal objetivo geral, os seguintes objetivos específicos de pesquisa foram trilhados:

- Estudo do estado da arte dos temas envolvidos: mineração de dados, redes bayesianas e sintonia de bancos de dados;
- Seleção do SGBD a ser utilizado para a criação do protótipo da ferramenta;
- Especificação e execução de experimentos no SGBD com distintos padrões de carga para geração de dados de treinamento;
- Elaboração, através de técnicas de mineração de dados, de um modelo probabilístico de identificação de regime de operação de um SGBD, capaz de identificar qual o padrão de carga sendo submetido a este SGBD;
- Elaboração de um modelo de otimização de parâmetros que afetam o desempenho do SGBD, através de experimentos de desempenho *(benchmarks)* executados em laboratório para as diferentes configurações para estes parâmetros;
- Projeto e desenvolvimento do protótipo da ferramenta de sintonia, baseado em sistema multiagente, capaz sugerir melhorias para a sintonia do SGBD.

O resultado final do trabalho é uma ferramenta que, através da monitoração e normalização dos registros (*log*) que indicam a carga de trabalho que está sendo submetida a um determinado banco de dados, enquadra esta carga em um particular regime de operação por meio de um modelo probabilístico de decisão. Os regimes de operação tratados neste trabalho correspondem às categorias de transações mais frequentemente executadas em SGBDs. Além disso, o trabalho também prevê a definição de um modelo de otimização, capaz de escolher os valores dos parâmetros de operação de SGBD que permitam otimizar seu desempenho. A escolha destes parâmetros é baseada na identificação prévia do regime de operação do SGBD.

1.3 Metodologia

Esta pesquisa envolve e integra diversas áreas da Ciência da Computação para solucionar o problema exposto. O projeto da ferramenta *ATTuneDB* depende de várias técnicas recentes de projeto de agentes autônomos, mineração de dados e algoritmos de treinamento.

Além disso, ao se trabalhar com a sintonia de um SGBD, alguns requisitos são prioritários. Desempenho, segurança e disponibilidade são os principais requisitos envolvidos no processo de sintonia. A presente pesquisa está limitada ao item desempenho, utilizando medidas de desempenho como norteadores para alcançar soluções que permitam ao usuário a configuração de um SGBD de forma que o desempenho esteja otimizado.

O projeto baseia-se no estado da arte em assuntos como mineração de dados, agentes autônomos, algoritmos de treinamento, modelos probabilísticos (bayesianos) de classificação, além de conceitos a respeito da arquitetura de SGBD. A exploração do estado da arte implica, além do estudo da literatura recente sobre estes temas, que se faça um estudo sobre quais os parâmetros de configuração que podem afetar o desempenho de um banco de dados. Estes serão os parâmetros cujos valores ideais serão sugeridos pela ferramenta. Este processo de definição de parâmetros também está associado à seleção do SGBD a ser usado na pesquisa. A própria seleção do SGBD também depende de aspectos como a disponibilidade de licença de uso do produto, de existência de ferramentas de monitoração de desempenho e do tipo de parâmetros de operação disponíveis no SGBD.

O método básico para a obtenção dos dados necessários para o treinamento do modelo

probabilístico de sintonia envolve o uso extensivo de experimentos controlados de laboratório. Este método também servirá de base para a validação do modelo e da ferramenta de otimização.

Embora a obtenção de dados de um SGBD em produção (operação comercial) seja mais realista, o fato é que existem grandes dificuldades práticas e de segurança na obtenção dos registros de *log* do mesmo. Até mesmo a ativação de mecanismos de geração de registros de *log* neste tipo de SGBD pode ser bastante restrita ou não permitida, por questões de desempenho. Por outro lado, SGBDs em produção usualmente apresentam regimes mistos de trabalho, tornando difícil a utilização destes dados para o treinamento de modelos de identificação do regime de trabalho. A condução de experimentos ou testes em SGBDs em produção usualmente também não é permitida. Uma alternativa possível seria a utilização de simuladores de SGBDs para a condução dos experimentos e geração dos dados. Porém, no presente trabalho considerou-se que essa alternativa não oferece as possibilidades de realismo que um método baseado em experimentação controlada com SGBDs reais, que são utilizados comercialmente em sistemas de produção, poderia oferecer.

Para que seja possível realizar os experimentos com o SGBD, é necessário desenvolver uma ferramenta de apoio cuja função será executar experimentos controlados de carga de trabalho no SGBD, a fim de medir o desempenho fornecido pelo SGBD com as diferentes configurações de parâmetros, para diferentes tipos de cargas. Os dados coletados nestes experimentos serviram para treinar o modelo de identificação do regime de trabalho do SGBD, necessário para a ferramenta de sintonia.

Na criação da ferramenta de sintonia, foram utilizadas técnicas de mineração de dados para se criar um modelo capaz de decidir, a partir dos registros de *log* que indicam que carga está sendo submetida ao banco, quais os valores que podem fazer parte de uma configuração que otimiza o desempenho do SGBD. Em se tratando das técnicas de mineração de dados empregadas, foram comparados vários algoritmos de treinamento, gerando diferentes tipos de modelos de classificação. Após esta comparação foi selecionado um algoritmo de classificação bayesiano. Através da utilização do classificador bayesiano, a ferramenta teve condições de inferir qual a categoria (classe) de comandos submetidos ao banco (carga) que mais se aproxima da carga real coletada, e a partir deste momento determinar o conjunto de parâmetros ideal, previamente encontrado através de um processo de pesquisa do espaço de configurações do SGBD baseado em um amplo conjunto de experimentos de desempenho.

O projeto do protótipo da ferramenta de sintonia foi baseado em uma arquitetura multiagente através do laço de controle autonômico. Em termos concretos, as tecnologias de

projeto de agentes autônomos forneceram a base para a autonomia da ferramenta, necessária para a detecção automática do que está ocorrendo com o SGBD. O modelo de decisão dos agentes que compõem a ferramenta está fundamentado em modelos bayesianos, permitindo o tratamento das incertezas inerentes a este processo de decisão.

1.4 Organização deste Trabalho

O texto está organizado em seções de maneira a evoluir o assunto e cobrir o trabalho que foi realizado, mostrando passo a passo como a solução foi construída e o problema foi atacado.

O capítulo 2 traz a fundamentação teórica a respeito dos temas envolvidos nesta pesquisa, começando com o tema mais abrangente e especializando-se em tópicos pontuais da pesquisa. O capítulo 3 apresenta os trabalhos relacionados a presente pesquisa. O capítulo 4 apresenta o processo de criação do modelo de identificação de regime de trabalho de SGBD que será utilizado pela ferramenta de sintonia. O capítulo 5 apresenta o modelo de otimização de parâmetros do SGBD. O capítulo 6 descreve as características do protótipo da ferramenta de sintonia, mostrando sua arquitetura, seu processo de desenvolvimento e de utilização dos modelos criados para resolver o problema de otimização do SGBD. O capítulo 7 mostra os resultados obtidos com a ferramenta através da aplicação do protótipo em um ambiente controlado. Por fim, o capítulo 8 apresenta as conclusões sobre os resultados alcançados, além de identificar novas possibilidades de pesquisa.

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo traz um embasamento teórico a respeito dos temas tratados nesta pesquisa. Começando com o tema mais abrangente a ser tratado, a seção 2.1 apresenta as principais características dos SGBDs, descrevendo os conceitos básicos, a finalidade e a organização desse tipo de sistema. A seção 2.2 discorre a respeito de sintonia de bancos de dados, uma técnica para que se tenha uma melhoria no seu desempenho. A seção 2.3 apresenta as características dos modelos probabilísticos baseados em redes bayesianas, incluindo conceitos elementares, aplicações, e funcionamento das mesmas. Na seção 2.4 são apresentas as principais características da tecnologia de mineração de dados. A seção 2.5 detalha os aspectos da tecnologia de sistemas autônomos baseados em agentes, e a última seção, 2.6, discorre a respeito da computação autonômica.

2.1 Sistemas Gerenciadores de Bancos de Dados

Sistemas Gerenciadores de Bancos de Dados são aplicações muito úteis e vastamente utilizadas. Os SGBDs ajudam, por exemplo, uma empresa a organizar os seus dados de forma concisa e organizada.

De acordo com Date (2000), um SGBD pode ser visto como um meio computadorizado de se armazenar registros. Além deste armazenamento de registros em meio eletrônico, um SGBD ainda possui outras funções, entre as quais permitir ao usuário buscar e atualizar estes dados quando necessário.

> Um sistema gerenciador de banco de dados (SGBD) é uma coleção de dados inter-relacionados e uma coleção de programas para acesso a estes dados. O conjunto de dados, comumente chamado banco de dados, contém informações sobre uma empresa em particular. O objetivo de um SGBD é proporcionar um ambiente tanto conveniente quanto eficiente para recuperação e armazenamento de informações do banco de dados (SILBERSCHATZ, FORTH e SUDARSHAN, 1999, p.1).

Os SGBDs estão presentes em computadores dos mais diversos portes. Quanto mais

potente o computador, mais recursos um SGBD instalado no mesmo conseguirá prover (DATE, 2000).

Uma das maiores vantagens de se utilizar um SGBD está no fato de que o mesmo proporciona um controle centralizado dos dados de uma instituição. Isto muitas vezes se vê no dia-a-dia das empresas, onde muitas aplicações armazenam os seus dados de forma privada e sem contato com o mundo exterior (DATE, 2000).

Apesar de todo o poder de processamento de um banco de dados, a atual exigência de desempenho dos mesmos e a necessidade de um SGBD cada vez mais eficiente e veloz faz com que muitos fabricantes permaneçam continuamente à procura de soluções para serem embutidas em seus produtos.

A figura 2.1 ilustra as partes que compõem um SGBD, mostrando como as mesmas se relacionam:

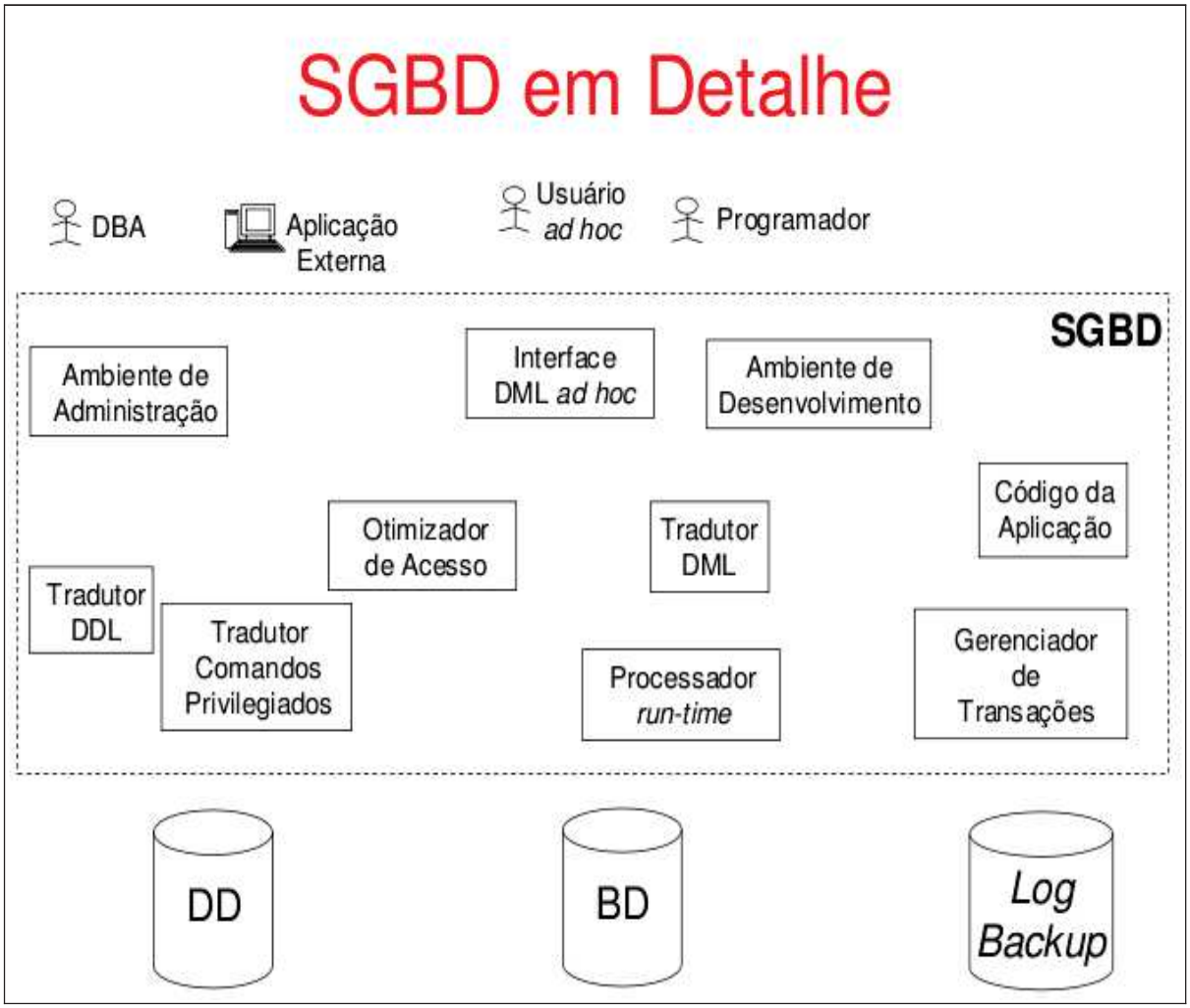

Figura 2.1: Detalhamento em alto-nível das partes relacionadas a um SGBD.
Fonte: SGBD (2010).

2.2 Sintonia de SGBDs

De acordo com Shasha e Bonnet (2003), a sintonia de banco de dados (*database tuning)* é a atividade de tornar a execução de uma aplicação de banco de dados mais veloz. Para a auto-otimização, é necessário maximizar a alocação de recursos e a sua utilização para satisfazer os requisitos do usuário com a mínima intervenção possível. Os componentes que usufruem da auto-otimização podem, em um segundo momento, aprender com os resultados passados, e então ajustar-se automaticamente para conseguir um desempenho ainda melhor do que o obtido anteriormente. Este ajuste deve ser transparente ao usuário, e a otimização deve sempre considerar políticas de alto nível definidas pelo mesmo.

SGBDs autonômicos têm recebido grande atenção dos meios acadêmico e comercial. Conceitos de auto-sintonia têm sido aplicados em problemas como seleção de índices, seleção de visões materializáveis e gerenciamento de memória (POWLEY et al., 2005).

De acordo com Date (2000), um SGBD se utiliza das diferentes partes de um computador para a sua operação. Acesso a registradores da CPU, acesso ao disco, entre outros, fazem parte das operações rotineiras de trabalho de um SGBD. Um estudo mostrado em Momjian (2001) define as diferentes áreas de armazenamento de um computador, como pode ser observado na figura 2.2.

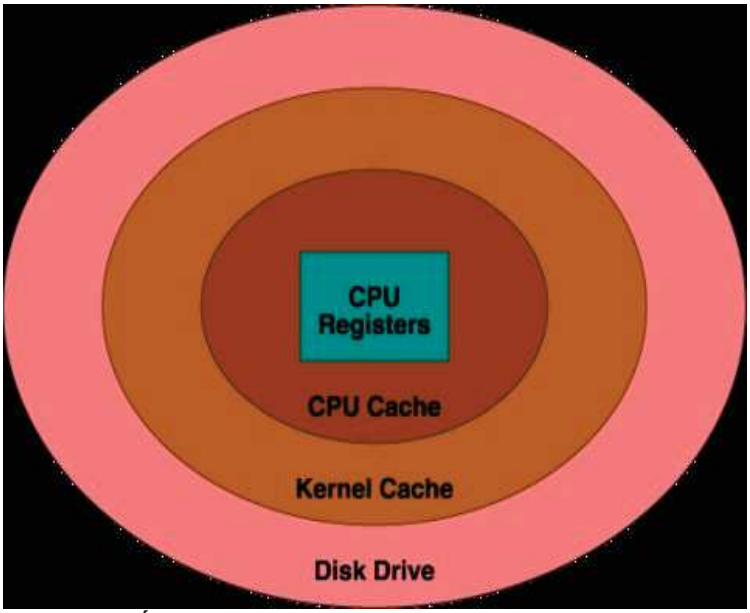

Figura 2.2: Áreas de armazenamento.
Fonte: Momjian (2001)

De acordo com Momjian (2001), quanto mais o dado se encontra nos componentes mais ao centro da imagem da figura 2.2, mais rápido o acesso à informação. Em Momjian (2001) é sugerido que, para que um SGBD possa ser otimizado de maneira eficiente, é interessante fazer com que a informação esteja o mais próximo possível da CPU (centro da imagem). Sabido isto, Momjian (2001) ainda sugere que os SGBDs mais atuais e conhecidos fornecem um conjunto de parâmetros configuráveis, os quais otimizam o acesso à informação, fazendo com que o SGBD tenha um desempenho otimizado.

O Quadro 2.1 demonstra alguns parâmetros de configuração do PostgreSQL (arquivo postgresql.conf), que possuem relação direta com o desempenho do SGBD. Alterações nos valores destes parâmetros configuram a sintonia do SGBD e o mesmos provêem ao SGBD um melhor desempenho quando configurados de maneira correta.

Item	Descrição
checkpoint_segments	Número máximo de segmentos de arquivos de *log* entre *checkpoints* WAL (*Write Ahead Log*) automáticos (cada segmento geralmente equivale a 16mb).
checkpoint_timeout	Tempo máximo entre *checkpoints* WAL automáticos, em segundos.
Shared_buffers	Esta configuração define a quantidade de memória que o servidor do banco de dados utiliza para *buffers* de memória compartilhada. O valor padrão é, geralmente, 32 megabytes (32MB), porém pode ser menor se as configurações do *kernel* do sistema operacional não suportam o valor acima.
effective_cache_size	A quantidade de memória RAM que será utilizada para *cache* efetivo do banco de dados. Esta configuração, na prática, faz com que o SGBD não precise de constantes leituras de tabelas e índices a partir do disco, mantendo-os em memória, visto o acesso ao disco ser mais custoso (LINUX DEPOT, 2008). O valor padrão desta configuração é de 128 megabytes (128MB).
wal_buffers	O número total de *buffers* utilizado pelo WAL. O WAL garante que os registros sejam gravados em *log* para possível recuperação antes de fechar uma transação (LINUX DEPOT, 2008).
commit_delay	O tempo de espera entre a gravação de um registro ao qual foi dado *commit* para o *buffer* WAL e a liberação do *buffer* para o disco, em microssegundos.

Quadro 2.1: Relação de Parâmetros de Configuração do PostgreSQL
Fonte: PostgreSQL (2010a); Duan, Thummala e Babu (2009).

A correta sintonia dos parâmetros do SGBD PostgreSQL permite uma notável melhora no desempenho do SGBD. No referido SGBD, a configuração dos parâmetros é feita através de um arquivo chamado *postgresql.conf*, o qual associa parâmetros a valores. A cada parâmetro, um valor associado é atribuído para que o SGBD considere. Para alguns parâmetros, é necessário que o SGBD seja reiniciado para que se carregue a nova configuração.

2.3 Modelos Probabilísticos Baseados em Redes Bayesianas

Não são raras as situações onde a falta de informações leva a situações de incerteza. Decisões podem deixar de ser tomadas por falta de insumos necessários à tomada das mesmas.

A principal vantagem do raciocínio probabilístico sobre raciocínio lógico é o fato de que agentes podem tomar decisões racionais mesmo quando não existe informação suficiente para se provar que uma ação funcionará (CHARNIAK, 1991).

Lidar com falta de informação significa lidar com incertezas. Nestes ambientes é necessário utilizar conectivos que manipulem níveis de certeza e não apenas valores booleanos, 1 e 0 (DUTRA, 2010).

Desta maneira, o trabalho com redes bayesianas nos permite representar situações do tipo:

a) Há uma probabilidade de 0,8 de que hoje chova;

b) Dado que choveu, há uma probabilidade de 0,95 de o chão estar molhado;

c) Dado que o chão está molhado, há uma probabilidade de 0,8 de ter chovido.

Estes são apenas alguns exemplos simples da utilização de incertezas, as quais podem ser trabalhadas através de redes bayesianas.

De acordo com Russel e Norvig (2003), o formalismo das redes bayesianas apareceu para facilitar a representação eficiente e a decisão rigorosa em situações nas quais se dispunha somente de conhecimento incerto.

2.3.1 Estrutura Básica e Representação

De acordo com Russel e Norvig (2003) uma rede bayesiana é um tipo de modelo gráfico cujos elementos são nós, setas entre os nós, e atribuições de probabilidades. Um conjunto finito de nodos, em conjunto com um conjunto de setas (ligações direcionadas) entre os nodos formam uma estrutura matemática chamada de grafo direcionado:

- Os nodos representam variáveis aleatórias, onde cada variável aleatória pode ser discreta ou contínua.
- Os arcos representam a relevância entre as variáveis; para cada variável X com os nodos pais Y1, Y2, …, Yn, há uma tabela de probabilidades condicionais associada, Pr(X |, Y1, Y2, …, Yn, I), onde I denota um conhecimento prévio, sabido, que é todo o conhecimento relevante que não aparece explicitamente sob a forma de nodos no grafo.

Uma sequência consecutiva de setas conectando dois nodos X e Y, independente da direção das setas, é conhecida como um caminho entre X e Y. Por exemplo, na figura 2.3 há dois caminhos diferentes ligando os nodos A e C.

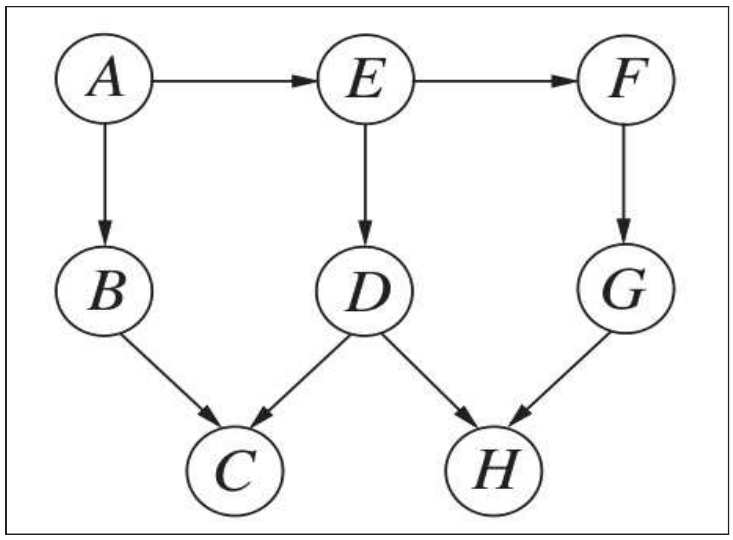

Figura 2.3: Grafo Direcionado.
Fonte: Russe e Norvig (2003).

De acordo com Castillo, Gutiérrez e Hadi (1998), quando se trata do problema de aprendizado das redes bayesianas, é importante notar que diferentes grafos dirigidos podem

representar as mesmas estruturas de independência e/ou as mesmas distribuições conjuntas para o conjunto de variáveis dado. Em outras palavras, quando se resolve o problema de aprendizagem no conjunto de todas as redes bayesianas, diversas soluções podem ser encontradas, mas algumas destas soluções representam as mesmas estruturas de independência e/ou as mesmas distribuições conjuntas.

De acordo com Cooper (1990), a computação de uma rede bayesiana é algo complexo, sendo considerado um problema NP-Hard, inclusive para redes cuja topologia pode ser significativamente restringida.

Apesar disto, para alguns tipos de grafos, chamados árvores de junção, há algoritmos eficientes para efetuar a computação das distribuições de probabilidade dos nodos exatamente (RUSSEL e NORVIG, 2003). Estes algoritmos permitem ao usuário a manutenção de problemas tanto com variáveis aleatórias contínuas quanto discretas.

2.3.2 Independência Condicional

De acordo com Russel e Norvig (2003), o conceito fundamental para a construção de redes bayesianas é a independência condicional. Geralmente ocorre que é necessário um julgamento a respeito do fato de uma dada proposição B ser relevante para uma outra proposição A em um contexto que inclui mais do que conhecimento do problema.

De acordo com Tan, Steinbach e Kumar (2009), supondo que X, Y e Z denotem três conjuntos de variáveis aleatórias, as variáveis de X são ditas condicionalmente independentes de Y, dado Z, se a seguinte condição for verdadeira: $P(X|Y,Z)=P(X|Z)$.

Segundo Tan, Steinbach e Kumar (2009), um exemplo de independência condicional é o relacionamento entre o comprimento do braço de uma pessoa e sua capacidade de leitura. Poderia ser observado que as pessoas com braços mais compridos tendem a ter maior capacidade de leitura, mas uma criança, por exemplo, tende a ter braços mais curtos e menor capacidade de leitura. No entanto, entra em cena o fator de confusão, que é a idade. Desta maneira, podemos concluir que o comprimento do braço e a capacidade de leitura são condicionalmente independentes quando a variável idade é fixa.

2.4 Mineração de Dados

De acordo com Tan, Steinbach e Kumar (2009), a mineração de dados é o processo de descoberta automática de informações úteis em grandes depósitos de dados. As técnicas de mineração de dados são organizadas para agir sobre grandes bancos de dados com o intuito de descobrir padrões úteis e recentes que poderiam, de outra forma, permanecer ignorados.

A mineração de dados é uma parte integral da descoberta de conhecimento em bancos de dados (KDD – Knowledge Discovery in Databases), que é o processo geral de conversão de dados brutos em informações úteis (TAN, STEINBACH e KUMAR, 2009). Este processo consiste de uma série de passos de transformação, indo do pré-processamento dos dados até o pós-processamento dos resultados da mineração de dados.

A figura 2.4 ilustra as etapas do processo de descoberta de conhecimento.

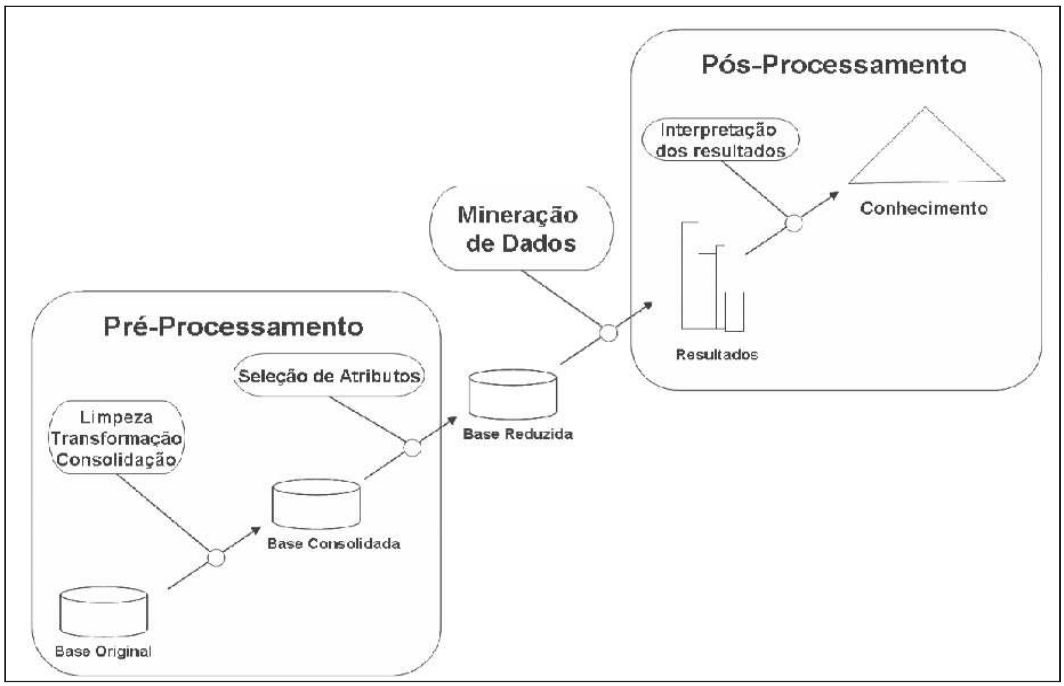

Figura 2.4: Etapas do processo de descoberta do conhecimento
Fonte: Lopes (2007)

Em Wei et al (2005), os autores discutem o uso da técnica de mineração de dados na área médica, e mostram o desenvolvimento de uma ferramenta para visualizar e trabalhar técnicas de mineração de dados sobre séries de dados de domínio da área da medicina. Os

autores mostram como o uso da mineração de dados pode ser eficiente para, por exemplo, um cardiologista encontrar uma contração ventricular prematura em um paciente. Os autores demonstram como a visualização de similaridades e de diferenças em uma coleção de imagens pode fazer com que estas conclusões sejam alcançadas.

Em Chatzidimitriou e Symeonidis (2009), os autores discutem os benefícios do uso de técnicas de mineração de dados no âmbito de cadeias de fornecimento (*supply chain*), através de agentes que podem fazer com que os *stakeholders* (figuras envolvidas no processo) possam maximizar o seu lucro, o qual representa um esforço permanente no momento da negociação entre dois agentes, vista a natureza competitiva do ambiente onde os mesmos se encontram, e também os interesses de cada um.

Em Han e Chang (2002), são discutidos os benefícios que a técnica de mineração de dados proporciona aos mecanismos de busca e compreensão de conteúdo de objetos HTML, XML e dados em um SGBD, armazenados na *web*. Para estabelecer uma ordenação (*ranking*) entre estes recursos, os pesquisadores aplicaram técnicas de mineração de dados, discutindo a crescente importância desta tecnologia para que se possa alcançar os desafios que surgem na tentativa de obtenção de uma *web* inteligente.

Na área de negócios, a mineração de dados pode ser utilizada em conjunto com a coleta de dados em pontos de venda, por exemplo, permitindo que se dê apoio a uma ampla gama de aplicações de inteligência de negócios como a criação de perfis de clientes, vendas direcionadas e a administração do fluxo de trabalho (TAN, STEINBACH e KUMAR, 2009).

Também nas áreas de ciências e engenharia, devido ao tamanho e a natureza espaço-temporal, métodos tradicionais não são apropriados para analisar conjuntos de dados massivos que são coletados a partir de satélites, por exemplo. As técnicas de mineração de dados podem ajudar os cientistas a obter respostas a perguntas acerca de desastres naturais, por exemplo (TAN, STEINBACH e KUMAR, 2009).

Percebe-se assim a grande variedade de aplicações que a mineração de dados pode ter. Devido à sua crescente importância, alguns dos SGBDs mais atuais fornecem recursos próprios para que se trabalhe com mineração de dados. Estes recursos são bastante interessantes à medida que o conhecimento pode ser extraído a partir dos dados armazenados nestes bancos. Os recursos fornecidos pelos SGBDs Oracle e SQLServer, por exemplo, demonstraram possuir uma usabilidade mais simples quando comparados com ferramentas comerciais conhecidas (CBMS, 2005). Em relação aos SGBDs citados acima, destacam-se as três etapas descritas a seguir, utilizadas por ambos os SGBDs para a mineração de dados.

Na etapa de criação do modelo, o usuário especifica o tipo de problema que deseja

> resolver através do emprego da mineração de dados. Podem ser formulados problemas como “qual o perfil do cliente que gasta muito em suas compras?” ou “quais os produtos cuja venda está mais relacionada?”. Após especificar os problemas que serão atacados, o usuário seleciona as tabelas do banco que deseja investigar, assim como os seus atributos. Na etapa seguinte, denominada treinamento do modelo, o usuário comanda a execução do algoritmo de mineração de dados sobre as tabelas selecionadas na etapa anterior. Como resultado, gera-se, de forma automática, um conjunto de regras e padrões extraídos destas tabelas. Nesta etapa, o usuário tem a liberdade de determinar qual a fatia dos dados que será examinada. No caso da loja de departamentos, tem-se, por exemplo, que o usuário poderia informar ao sistema que apenas os dados referentes às vendas do mês de novembro de 2006 deveriam ser investigados. Por fim, na etapa de consulta dos resultados, o usuário tem condições de explorar o conjunto de regras e padrões descobertos pelo algoritmo de mineração de dados. Estas regras são disponibilizadas de maneira simples, através de uma visão (view) do banco de dados, por exemplo. O usuário pode consultar esta visão, para avaliar todos os padrões descobertos (WEI et al., 2005, p.1).

Em resumo, a mineração de dados também pode ser vista como uma forma de selecionar, explorar e modelar grandes conjuntos de dados para detectar padrões de comportamento (FAYYAD, PIATETSKY-SHAPIRO e SMYTH, 1996).

2.4.1 Classificação de Dados

De acordo com Tan, Steinbach e Kumar (2009), os dados de entrada da tarefa de classificação são um conjunto de registros. Cada registro, também conhecido como uma instância ou exemplo, é caracterizado por uma dupla (x,y) onde x é o conjunto de atributos e y o atributo especial, designado rótulo de classe (também chamado de atributo alvo ou de categorização).

Desta forma, a classificação é caracterizada como sendo a tarefa de aprender uma função alvo que mapeie cada conjunto de atributos para um dos rótulos de classes pré-determinados.

Ainda de acordo com Tan, Steinbach e Kumar (2009), uma técnica de classificação é uma abordagem sistemática para construção de modelos de classificação a partir de um conjunto de dados de entrada. Exemplos incluem classificadores de árvores de decisão, classificadores baseados em regras, redes neurais, máquinas de vetor de suporte e classificadores bayesianos simples *(naive-bayes)*. As diferentes técnicas citadas utilizam diferentes algoritmos de aprendizagem para identificar um modelo que seja mais apropriado para o relacionamento entre o conjunto de atributos e o rótulo da classe dos dados de entrada.

A figura 2.5 ilustra uma abordagem geral para a construção de um modelo de classificação.

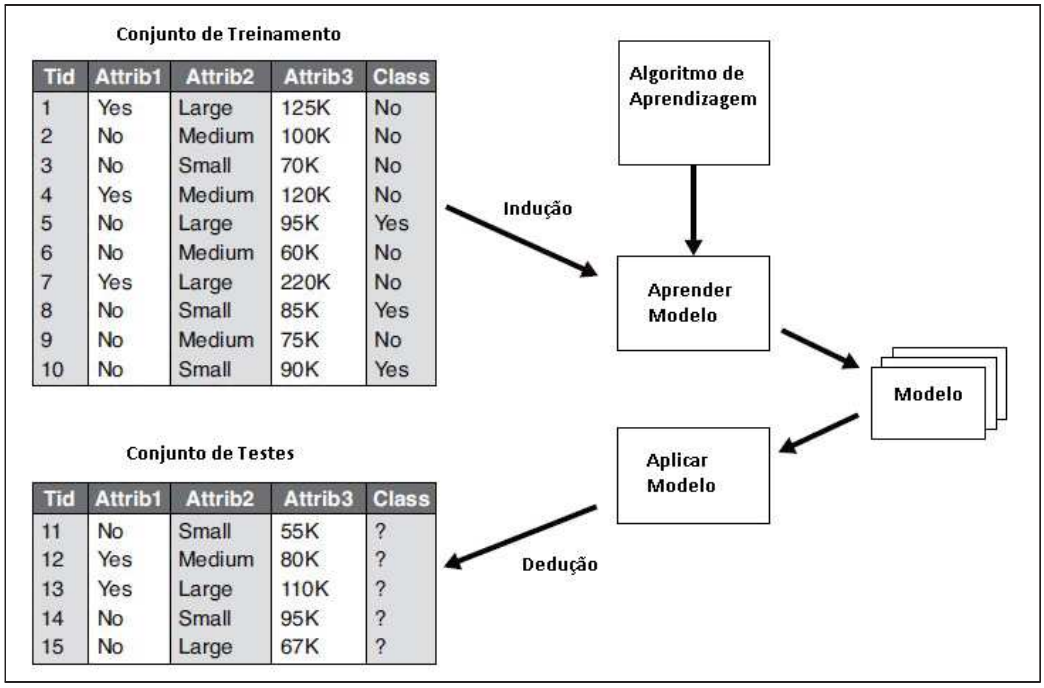

Figura 2.5: Abordagem geral para construção de um modelo de classificação (adaptado)
Fonte: Tan, Steinbach e Kumar, 2009.

Tem-se primeiramente um conjunto de treinamento, o qual possui os registros que são fornecidos ao modelo. Este conjunto é usado para a construção do modelo de classificação, sendo que este é aplicado ao conjunto de teste, o qual consiste de registros com rótulos de classes desconhecidos (TAN, STEINBACH e KUMAR, 2009).

De acordo com Lopes (2007) (apud Carvalho (2001)), para se executar a tarefa de classificação, são usados dados que consistem em um conjunto de atributos denominados previsores, e um atributo denominado preditor (classe). Os atributos previsores são utilizados para definir uma classificação efetiva dos registros pertencentes à base de dados em estudo. O atributo preditor por sua vez é utilizado como uma hipótese de classificação que será validada ou não pela análise resultante da classificação por meio dos atributos previsores.

Sendo assim, um algoritmo de classificação usa uma base de dados dividida dois conjuntos de instâncias mutuamente exclusivas. Um dos subconjuntos é chamado conjunto de treinamento e o outro conjunto de teste. Inicialmente, o conjunto de treinamento é percorrido, analisando as relações existentes entre os atributos previsores e o atributo preditor. Estas

relações são então usadas para prever a classe dos registros presentes no conjunto de teste, que será a próxima ação do classificador (LOPES, 2007 apud MITCHELL, 1997).

2.4.2 Algoritmos de Aprendizagem

De acordo com Russel e Norvig (2003), há diversos algoritmos de aprendizagem que podem ser utilizados nas técnicas de classificação de dados. Nesse contexto, Mitchell (1997) descreve as técnicas de aprendizado de máquina, e categoriza algoritmos de acordo com o tipo de aprendizagem. Alguns tipos de aprendizagem estão na lista abaixo:

a) Aprendizagem de Conceito;
b) Aprendizagem de Árvores de Decisão;
c) Aprendizagem Baseada em Instâncias;
d) Aprendizagem Bayesiana;
e) Aprendizagem de Redes Neurais.

Alguns algoritmos de aprendizagem podem ser utilizados em conjunto com modelos probabilísticos, incluindo redes bayesianas. De acordo com Mitchell (1997), o pensamento bayesiano fornece uma abordagem probabilística para a aprendizagem, pois está baseado no fato que quantidades de interesse estão associadas a distribuições de probabilidades. Os algoritmos mais populares para a aprendizagem associada às redes bayesianas são o classificador bayesiano simples (*Naive-Bayes*) e o algoritmo de Gibbs.

De acordo com Tan, Steinbach e Kumar (2009), cada técnica de classificação utiliza um algoritmo de aprendizagem para identificar um modelo que seja mais apropriado para descrever o relacionamento entre o conjunto de atributos e o rótulo de classe dos dados de entrada. Para que isto aconteça, o modelo gerado pelo algoritmo de aprendizagem deve se adaptar bem aos dados de entrada e prever corretamente os rótulos de classes de registros que ele nunca viu antes.

2.4.3 Algoritmo de Classificação Bayesiano Simples (*Naive Bayes*)

Tan, Steinbach e Kumar (2009) descrevem algumas características inerentes a um classificador Bayes simples. Eles são robustos para pontos de ruídos isolados porque calculam a média de tais pontos ao avaliar probabilidades condicionais a partir de dados. Classificadores Bayes simples também podem lidar com valores que estão faltando ignorando o exemplo durante a construção e classificação do modelo. Eles são robustos para atributos irrelevantes. Se X for um atributo irrelevante, então P (X|Y) se torna quase que uniformemente distribuído. A probabilidade condicional de classe para X não tem impacto no cálculo geral da probabilidade posterior (TAN, STEINBACH E KUMAR, 2009).

Dada a complexidade dos classificadores bayesianos, é necessário que se busque maneiras de diminuí-la. O classificador Naive Bayes faz isto assumindo uma independência condicional que reduz drasticamente o número de parâmetros a serem estimados quando se modela (P X|Y). A complexidade original de um classificador bayesiano é definida por 2(2n − 1). O classificador Naive Bayes reduz esta complexidade para simplesmente 2n , através da independência condicional citada anteriormente (Mitchell, 1997).

De acordo co Braga e Lacerda (2004), a fórmula do classificador Naive Bayes é definida pela seguinte equação:

$$p(x|w_j) = \prod_{i=1}^{d} p(x_i|w_j)$$

De acordo com Zhang (2004), o classificador bayesiano simples é um dos mais eficientes e eficazes algoritmos de aprendizagem para aprendizado de máquina e mineração de dados. Seu desempenho competitivo em tarefas de classificação é surpreendente, por causa da independência condicional na qual está baseado.

Braga e Lacerda (2004) ilustram o classificador através de um exemplo, demonstrado a seguir e baseado nos dados demonstrados no quadro 2.2.

Exemplo	Tempo	Temperatura	Umidade	Vento	Classe
1	Ensolarado	Quente	Alta	Fraco	Não Jogar
2	Ensolarado	Quente	Alta	Forte	Não Jogar
3	Nublado	Quente	Alta	Fraco	Jogar
4	Chuva	Média	Alta	Fraco	Jogar
5	Chuva	Frio	Normal	Fraco	Jogar
6	Chuva	Frio	Normal	Forte	Não Jogar
7	Nublado	Frio	Normal	Forte	Jogar
8	Ensolarado	Média	Alta	Fraco	Não Jogar
9	Ensolarado	Frio	Normal	Fraco	Jogar
10	Chuva	Média	Normal	Fraco	Jogar
11	Ensolarado	Média	Normal	Forte	Jogar
12	Nublado	Média	Alta	Forte	Jogar
13	Nublado	Quente	Normal	Fraco	Jogar
14	Chuva	Média	Alta	Forte	Não Jogar

Quadro 2.2: Dados do Jogo de Tênis
Fonte: Braga e Lacerda, 2004.

A topologia da rede bayesiana inferida do conjunto de dados através do algoritmo de aprendizagem do classificador bayesiano simples assume uma estrutura padrão que pode ser vista na figura 2.6.

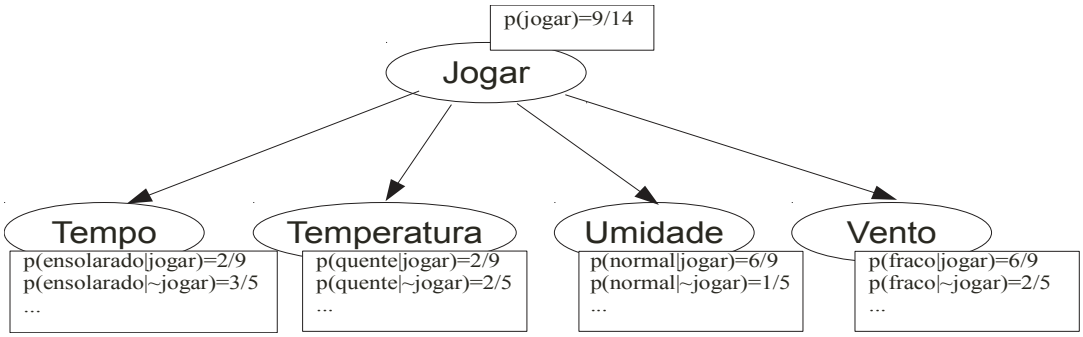

Figura 2.6: Rede Bayesiana Correspondente ao Classificador Bayesiano Simples

A partir da rede bayesiana definida na figura 2.6 pode-se aplicar a fórmula apresentada anteriormente, obtendo-se os resultados apresentados na figura 2.7.

$$p(\text{não jogar}|\text{ensolarado}, \text{quente}, \text{normal}, \text{fraco}) = p(\text{não jogar}) \times p(\text{ensolarado}|\text{não jogar}) \times p(\text{quente}|\text{não jogar}) \times p(\text{normal}|\text{não jogar}) \times p(\text{fraco}|\text{não jogar}) = \frac{5}{14} \times \frac{3}{5} \times \frac{2}{5} \times \frac{1}{5} \times \frac{2}{5} = 0.0069$$

$$p(\text{jogar}|\text{ensolarado}, \text{quente}, \text{normal}, \text{fraco}) = p(\text{jogar}) \times p(\text{ensolarado}|\text{jogar}) \times p(\text{quente}|\text{jogar}) \times p(\text{normal}|\text{jogar}) \times p(\text{fraco}|\text{jogar}) = \frac{9}{14} \times \frac{2}{9} \times \frac{2}{9} \times \frac{6}{9} \times \frac{6}{9} = 0.0141$$

Figura 2.7: Resultados da Aplicação da fórmula para a partida de Tênis
Fonte: Braga e Lacerda, 2004.

Dessa forma, conclui-se que, para o dia que se está buscando a resposta, é recomendável jogar tênis.

2.5 Sistemas Autônomos Baseados em Agentes

A modelagem e programação de sistemas baseados em agentes pode oferecer um caminho concreto para a implementação de sistemas autônomos. Agentes são entidades de *software* que incorporam os princípios de computação autonômica em seus conceitos mais básicos. Segundo Wooldridge (2002), agentes são justamente entidades de software situados em um determinado ambiente, capazes de perceber eventos e atuar sobre o ambiente, que são projetados para atingir determinados objetivos através de mecanismo de raciocínio prático. Apesar de existir outras definições para a noção de agentes e, mais genericamente, de agência, essa definição tem sido adotada por diversos autores recentes (BORDINI et al., 2007; RUSSEL e NORVIG, 2003; WEISS, 1999).

Além dessa definição é usual pressupor um conjunto de requisitos que separam o conceito de agente de outros conceitos mais usuais na computação, como processos ou

objetos de software. De acordo com Bordini et al. (2007), estes requisitos são os seguintes:

- *Autonomia*: capacidade do agente de tomar iniciativa e controlar suas próprias ações.
- *Reatividade*: capacidade que permite ao agente perceber o ambientes e responder adequadamente e em tempo hábil as mudanças nele ocorridas.
- *Pro-atividade*: capacidade do agente de ter seu comportamento dirigido a objetivos, podendo começar de maneira independente a execução das ações necessárias para atingir seus objetivos.
- *Sociabilidade*: capacidade de se comunicar (interagir) com outros agentes (ou seres humanos) para concluir suas tarefas.

Todas estas características são importantes para a concepção e projeto de sistemas autônomos: desde o princípio o agente é concebido como uma entidade autônoma, flexível e pró-ativa, inserida em um determinado meio. As percepções do agente são fornecidas por seus sensores, suas ações se refletem em movimentos de seus atuadores. Mecanismos de representação de conhecimentos são usados para representar as informações sobre o ambiente, sobre os objetivos do agente e sobre o estado de alcance desses objetivos. Modelos de decisão são usados para planejar e selecionar de forma autônoma o curso de ação (comportamento) do agente para atingir estes objetivos, revisando o planejamento quando necessário.

Já existe um conjunto considerável de técnicas de programação e construção de sistemas baseados em agentes (BORDINI et al., 2005) e de metodologias de engenharia de software apropriadas para o projeto de tais tipos de sistema (HENDERSON-SELLERS e GIORGINI, 2005), incluindo a definição de vários tipos de arquiteturas de agentes.

Uma arquitetura de agente que parece particularmente adequada ao presente trabalho é a arquitetura cognitiva BDI (*Beliefs - Desires – Intentions*) que é baseada nos estados mentais de crenças, desejos e intenções. Esta arquitetura é importante para a presente proposta, porque existem vários trabalhos recentes que mostram que ela pode ser combinada com mecanismos de inferência bayesiana (VICARI e GLUZ, 2007)(GLUZ et al., 2007)(VICARI et al., 2008).

BDI é uma arquitetura caracterizada por três estados mentais que são as crenças, os desejos e as intenções. As crenças representam tudo aquilo que o agente sabe sobre o ambiente e sobre os agentes daquele ambiente (inclusive sobre si mesmo). Os desejos representam os estados do mundo que o agente quer atingir. As intenções representam a seqüência de ações que um agente compromete-se a executar para atingir sua meta. Os principais criadores da arquitetura BDI foram Georgeff e Rao (BORDINI e VIEIRA, 2003). A

fundamentação filosófica para esta concepção de agentes vem do trabalho de Dennett (1987) sobre sistemas intencionais e de Bratman (1987) sobre raciocínio prático. O modelo da arquitetura BDI genérico é apresentado na figura 2.7.

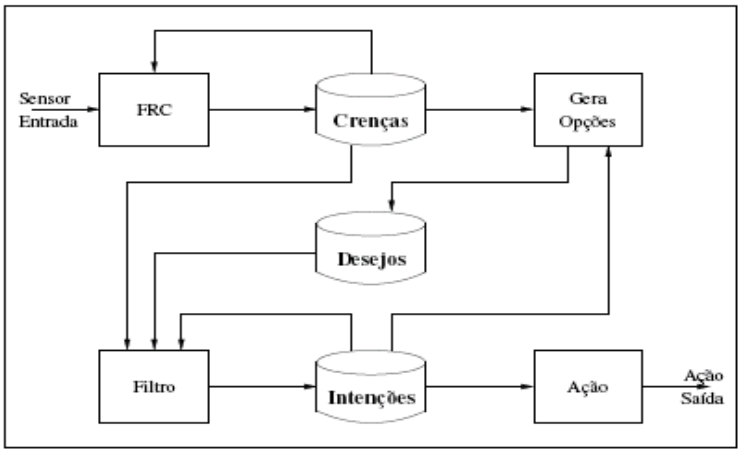

Figura 2.8: Arquitetura BDI genérica.
Fonte: Bratman, 1988.

No modelo BDI, uma crença é um estado mental que representa o conhecimento sobre o mundo no qual o agente está inserido. Do ponto de vista computacional, crenças são apenas maneiras de representar o estado do mundo, seja através de variáveis, uma base de dados relacional, ou expressões simbólicas em um cálculo de predicados. As crenças são consideradas essenciais, pois o mundo é dinâmico (eventos passados precisam ser lembrados), e os sistemas têm apenas uma visão local do mundo (eventos fora da sua esfera de percepção devem ser lembrados). Os desejos, por sua vez, estão relacionados ao estado de mundo que o agente quer atingir. Não necessariamente o fato de um agente possuir um desejo implica em agir para satisfazê-lo, significa que antes de um determinado agente decidir o que fazer, ele passa por um processo de racionalização e confronta os seus desejos com as suas convicções e, consequentemente, escolherá os desejos que são possíveis, de acordo com algum critério.

As intenções correspondem a estados de mundo que o agente quer efetivamente atingir. Intenções podem ser consideradas um subconjunto dos desejos, mas ao contrário destes, devem ser consistentes entre si. As intenções são formadas a partir de um processo de deliberação e a partir do refinamento de outras intenções. No entanto, um agente pode conter intenções iniciais inseridas pelo usuário. Normalmente, o termo intenção é empregado tanto para caracterizar um estado mental quanto para caracterizar uma ação. De acordo com

Wooldridge (2000) as intenções possuem as seguintes propriedades:

- Direcionam o raciocínio meio-fim: uma vez formada uma intenção, o agente deve tentar realizá-la através de um plano, caso um plano particular venha a falhar, o agente deve tentar outro;
- Persistência: uma intenção deve persistir até que seja realizada, e deve ser abandonada somente quando constatado que não é mais possível realizá-la, ou a razão que a formou deixou de existir;
- Restrição de deliberações futuras: o agente não selecionará novas intenções que são inconsistentes com as atuais;
- Influência sobre crenças utilizadas como base para raciocínios práticos futuros: um agente realiza planos futuros partindo do pressuposto que as suas intenções serão realizadas.

O processo deliberativo na arquitetura BDI consiste em formar as novas intenções do agente com base nas crenças, desejos e intenções atuais do mesmo. O processo normalmente é formado por duas etapas. Primeiramente, a *Geração de Opções,* que consiste na escolha de um conjunto opções (desejos) levando em conta as crenças e as intenções atuais do agente. E em seguida, a *Filtragem*, que tem como objetivo escolher a melhor alternativa gerada pela etapa anterior, formando assim a nova intenção.

2.6 Computação Autonômica

Hoje em dia, cresce-se em números. Tem-se mais softwares desenvolvidos, cria-se mais dispositivos de hardware*,* porém estima-se que em dez anos será necessária uma mão-de-obra equivalente à população dos Estados Unidos para se manter toda esta estrutura (MURCH, 2003). O principal obstáculo que está se impondo para a computação é a complexidade. Lidar com a complexidade é o desafio mais importante que a indústria da tecnologia da informação deve enfrentar nos próximos tempos (HORN, 2001).

> O paradigma da computação autonômica foi inspirado pelo sistema nervoso autonômico. Seu principal objetivo é desenvolver sistemas de software e aplicações que podem gerenciar a si próprios de acordo com políticas de alto nível descritas por humanos. Alcançar os grandes desafios da computação autonômica requer avanços científicos e tecnológicos em uma larga variedade de campos, bem como novos paradigmas de programação e software e arquiteturas de sistema que suportem a integração efetiva das tecnologias que a constituem (PARASHAR e HARIRI, 2007, p. 8).

Para se ajudar a contornar estes desafios, a computação autonômica surge com uma série de conceitos que podem ser implementados em sistemas computacionais com a finalidade de torná-los auto-gerenciáveis. Para um sistema ser auto-gerenciável, algumas características são necessárias. De acordo com Murch (2003), as principais características, também chamadas de *selfs,* são a *auto-configuração*, a *auto-otimização*, a *auto-proteção* e a *auto-cura*.

A figura 2.9 demonstra as quatro características autonômicas que podem estar presente em um sistema para que o mesmo seja autonômico. Estas características podem aparecer associadas ou de forma isolada, dependendo do contexto da aplicação em questão.

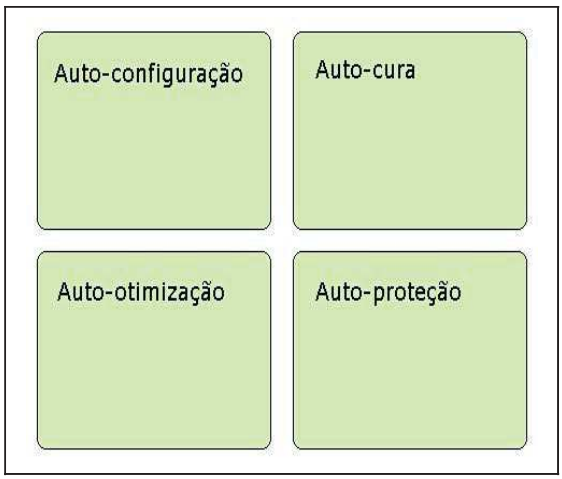

Figura 2.9: As Características Básicas de um Sistema Autonômico

2.6.1 Auto-configuração

A auto-configuração refere-se ao fato de que os sistemas devem adaptar-se

dinamicamente. Caso qualquer mudança ocorra no ambiente onde o sistema está inserido, o mesmo deve entender esta mudança e se re-configurar a tal ponto de estar pronto a tomar proveito destas novas configurações (MURCH, 2003).

Esta característica permite ao sistema adaptar-se a condições inesperadas mudando automaticamente as suas configurações. Para tanto, o sistema pode adicionar ou remover recursos ou também instalar algum componente adicional de maneira transparente ao usuário (PARASHAR e HARIRI, 2007). Todas estas operações efetuadas pelo sistema devem requerer uma mínima intervenção do usuário e poder ser efetuadas imediatamente. (MANOEL et al., 2005).

2.6.2 Auto-otimização

Um sistema auto-otimizável deve eficientemente efetuar melhorias em itens internos, como, por exemplo, o uso correto de recursos disponíveis, e otimizar-se automaticamente (MURCH, 2003), de maneira a prover um funcionamento otimizado em relação ao estado em que se encontrava no momento anterior. A auto-otimização é uma das características autonômicas mais importantes, e sua reflexão é facilmente percebida.

Esta característica permite ao sistema melhorar seu desempenho permanentemente, tanto em processos já existentes quanto em respostas a condições e interações com o ambiente (PARASHAR e HARIRI, 2007).

Para a auto-otimização, é necessário maximizar a alocação de recursos e a sua utilização para ir de encontro às necessidades do usuário com a mínima intervenção possível (MANOEL et al., 2005). Os componentes que usufruem da auto-otimização podem, em um segundo momento, aprender com os resultados passados, e então ajustar-se automaticamente para conseguir um desempenho ainda melhor do que o obtido anteriormente. Este ajuste deve ser transparente ao usuário, e a otimização deve sempre considerar políticas de alto nível definidas pelo mesmo.

2.6.3 Auto-proteção

Todo sistema deveria estar preparado contra ataques maliciosos. A computação autonômica provê aos sistemas uma maneira de se proteger de maneira automática. Não é necessária a intervenção de um programador ou de um administrador, o sistema deve ser capaz de fazê-lo sozinho (MURCH, 2003). Entre os perigos que ameaçam os sistemas e que devem ser detectados automaticamente pelos sistemas autonômicos estão os ataques de vírus e os acessos sem autorização (PARASHAR e HARIRI, 2007). Um sistema que possui a característica da auto-proteção deve estar apto a detectar estas ameaças e as contornar, dispensando a interferência de um humano.

De acordo com Manoel et al. (2005), um ambiente dotado de auto-proteção deve permitir às pessoas autorizadas o acesso aos dados certos na hora certa, e automaticamente tomar as ações corretas para tornar-se menos vulnerável a ataques em sua infraestrutura.

2.6.4 Auto-cura

Se um sistema está prestes a entrar em um estado falho, o mesmo deve ser capaz de sair por si só da situação problemática. Esta capacidade é conhecida por auto-cura, e se refere ao fato de que um sistema precisa resolver situações críticas de maneira automática (MURCH, 2003).

Os principais pontos a serem tratados são a prevenção de problemas e a recuperação automática de situações problemáticas, itens que podem ser adquiridos através de serviços de descobertas, diagnósticos e recuperações de questões que podem causar falhas nas funcionalidades do sistema (PARASHAR e HARIRI, 2007).

A recuperação de uma situação de falha do sistema deve acontecer com total transparência ao usuário. Um ambiente que se auto-cura deve tomar as ações corretivas sem corromper as aplicações do sistema que estão sendo executadas, preservando a sua integridade (MANOEL et al., 2005).

2.6.5 Componentes de um Sistema Autonômico

Desenvolver um sistema com características autonômicas é uma tarefa difícil. Para amenizar este processo, foi proposta pela IBM uma arquitetura representada por um laço de controle, chamada de laço de controle autonômico.

O laço de controle autonômico sugere que sejam divididos os componentes do sistema autonômico em categorias pré-estabelecidas. Estas categorias interagem entre si formando o laço.

De acordo com Manoel et al. (2005), em um ambiente autonômico os componentes trabalham juntos, comunicando-se um com o outro e ainda com ferramentas de gerenciamento de alto nível. Eles podem gerenciar a si mesmos, ou então gerenciar um ao outro.

Através deste laço, um gerenciador autonômico monitora os detalhes dos recursos, os analisa, planeja ajustes e executa os ajustes planejados, utilizando informações humanas e regras definidas por usuário ou aprendidas pelo sistema. (PARASHAR e HARIRI, 2007).

A Figura 2.10 demonstra o laço de controle autonômico e a maneira como os componentes trabalham e comunicam-se entre si.

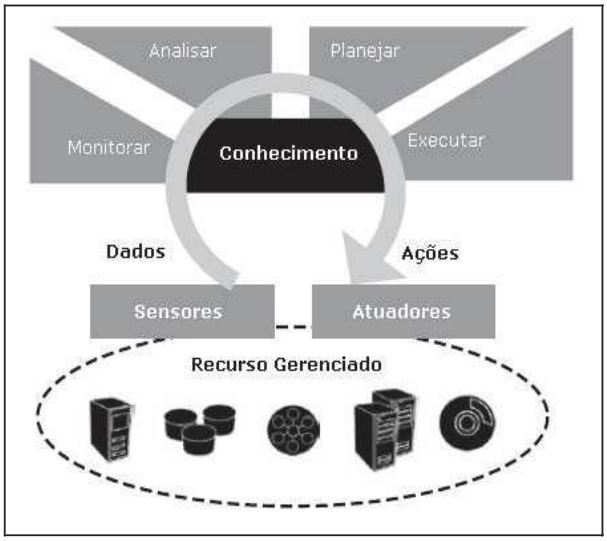

Figura 2.10: O Laço de Controle Autonômico
Fonte: PARASHAR e HARIRI (2007).

De acordo com Manoel et al. (2005), um monitor deve prover mecanismos que coletam, agregam, filtram, gerenciam, e reportam detalhes coletados de um elemento. Um

analisador provê mecanismos que correlacionam e modelam situações complexas. Estes mecanismos habilitam o gerenciador autonômico a aprender sobre o ambiente e ajudar a prever situações futuras (MANOEL et al., 2005). Um planejador provê os mecanismos que estruturam a ação necessária para se atingir os objetivos. O mecanismo de planejamento utiliza informações de regras previamente definidas para guiar o seu trabalho (JACOB et al., 2004). Um executor provê mecanismos que controlam a execução de um plano, sempre considerando atualizações que podem acontecer a qualquer momento (MANOEL et al., 2005).

Como pode ser observado na figura 2.10, o recurso gerenciado possui, acoplado a si, sensores e atuadores, que são os componentes responsáveis pela iteração desse recurso com o restante dos componentes envolvidos no sistema.

Os sensores permanecem junto ao recurso gerenciado coletando as informações necessárias para que o sistema funcione corretamente.

> Os sensores provêem mecanismos para coletar informações a respeito do estado e sobre a transição de estados de um elemento. Sensores podem ser implementados utilizando um conjunto de operações *get* para recuperar informações sobre o estado atual, ou um conjunto de eventos de gerenciamento (não-solicitados, mensagens assíncronas, ou notificações) que fluem quando o estado de um elemento muda em um caminho significativo, ou ambos (MANOEL et al., 2005, p. 8).

Os atuadores permanecem junto ao recurso gerenciado executando as operações solicitadas pelos outros componentes do sistema, a fim de garantir que estas operações sejam executadas no recurso gerenciado.

> Os atuadores são mecanismos que alteram o estado (configuração) de um elemento. Em outras palavras, os atuadores são uma coleção de comandos *set* ou interfaces de programação de aplicação (APIs) que alteram a configuração do recurso gerenciado de alguma maneira (MANOEL et al., 2005, p. 8).

A combinação de sensores e atuadores forma a interface de gerenciabilidade que está disponível para um gerenciador autonômico (JACOB et al., 2004). Esta arquitetura encoraja a idéia de que sensores e atuadores estão ligados.

De acordo com Parashar e Hariri (2007), o bloco que implementa o comportamento de sensores e atuadores para um ou mais mecanismos de um recurso gerenciado é chamado de

touchpoint.

Para Jacob et al. (2004), uma mudança de configuração que ocorre através de atuadores deve refletir como uma notificação de mudança de configuração através da interface de um sensor, demonstrando a ligação entre os dois componentes.

3 TRABALHOS RELACIONADOS

Este capítulo discorre acerca de trabalhos correlatos à presente pesquisa. A seção 3.1 fala sobre trabalhos referentes à sintonia de parâmetros de SGBDs para a melhoria de desempenho. A seção 3.2 mostra o atual estado das pesquisas com computação autonômica e a sua aplicação nos SGBDs atuais, bem como trabalhos próximos desenvolvidos na área. O uso de agentes para ganho de desempenho em SGBDs é discutido através dos trabalhos detalhados na seção 3.3. Por fim, a seção 3.4 faz uma análise dos trabalhos apresentados, os compara e demonstra em quais pontos a presente pesquisa se destaca em relação às demais.

3.1 Sintonia de Parâmetros de SGBDs para Melhoria de Desempenho

Tradicionalmente, os SGBDs conhecidos são vendidos ou disponibilizados com centenas de parâmetros para a sua configuração. O melhor conjunto de valores atribuídos a estes parâmetros depende de diversos fatores, como o ambiente computacional envolvido, a finalidade desta sintonia e a finalidade de uso do SGBD em questão.

Debnath, Lilja e Mokbel (2008) utilizaram-se de técnicas estatísticas para estabelecer uma ordenação (*ranking*) dos parâmetros de um SGBD que são mais significativos no momento de se efetuar uma sintonia fina. A ferramenta, denominada SARD, considera a carga que está sendo submetida ao banco de dados, bem como o número de parâmetros de configuração do SGBD como entrada para os experimentos. A partir de então, utiliza um número linear de experimentos para gerar uma ordenação de parâmetros do SGBD, baseada no impacto relativo que cada um tem sobre o desempenho deste SGBD para o nível de carga proposta.

Em geral, os SGBDs possuem, em sua instalação padrão, valores padrão para os seus parâmetros. Apesar de alguns SGBDs possuírem ferramentas para efetuar a recomendação de valores para estes parâmetros (como o DB2 autoconfigure, por exemplo), os valores sugeridos não consideram a carga que está sendo submetida ao SGBD. Os autores ainda esclarecem que, além disto, a correta configuração dos parâmetros envolvidos na sintonia do SGBD depende das habilidades de um DBA, o qual é uma figura cada vez mais rara e cara no mercado.

A ideia principal da ferramenta SARD é conduzir um conjunto de experimentos que

utilizam uma amostragem aproximada do espaço total de valores possíveis para cada um dos parâmetros utilizados nos testes. Em cada teste, os valores dos parâmetros são modificados sistematicamente através de um conjunto de valores aceitos para o referido parâmetro.

O quadro 3.1 exibe alguns parâmetros que foram utilizados pela ferramenta bem como os valores utilizados para estes parâmetros em cada teste.

Parâmetro	Valor 1	Valor 2
effective_cache_size (páginas)	81920	8192
maintenance_work_mem (páginas)	8192	1024
shared_buffers (páginas)	16384	1024
temp_buffers (páginas)	8192	1024
work_mem (KB)	8192	1024
checkpoint_timeout (segundos)	1800	60
deadlock_timeout (milisegundos)	60000	100
max_connections	5	100

Quadro 3.1: Alguns Valores Utilizados para os Parâmetros Trabalhados
Fonte: Debnath, Lilja e Mokbel (2008)

Cada experimento representa uma específica sintonia do SGBD, pois é executado com um conjunto associado de diferentes valores para os parâmetros envolvidos. Uma análise subsequente dos resultados encontrados é realizada para que se encontre os efeitos dos parâmetros no desempenho do sistema.

A contribuição deste trabalho é, principalmente, uma metodologia para estabelecer uma ordenação dos parâmetros de configuração de um SGBD baseada em seu impacto sobre o desempenho para uma dada carga de trabalho.

Uma avaliação preliminar dos resultados encontrados mostra que a ferramenta SARD foi capaz de encontrar os parâmetros de gargalo do desempenho, utilizando o SGBD PostgreSQL em conjunto com o *benchmark* TPC-H (TPC-H, 2010).

Este trabalho também valida o fato de que a mudança nos valores dos parâmetros no SGBD realmente impacta em seu desempenho, ou seja, uma correta configuração de valores para estes parâmetros de fato faz com que o SGBD tenha o seu desempenho melhorado.

Considerando o impacto, para uma sintonia fina, da alteração de valores dos parâmetros de um SGBD, Machado e Nascimento (2008) demonstraram o ganho de

desempenho através de um agente autonômico que, através da manipulação dos valores dos parâmetros de um SGBD PostgreSQL, realiza testes empíricos em horários pré-definidos para encontrar o melhor desempenho efetivamente alterando os valores dos parâmetros e reavaliando o desempenho a cada alteração.

O fluxo de execução do agente ADA (*Autonomic Database Agent)* desenvolvido é demonstrado na figura 3.1.

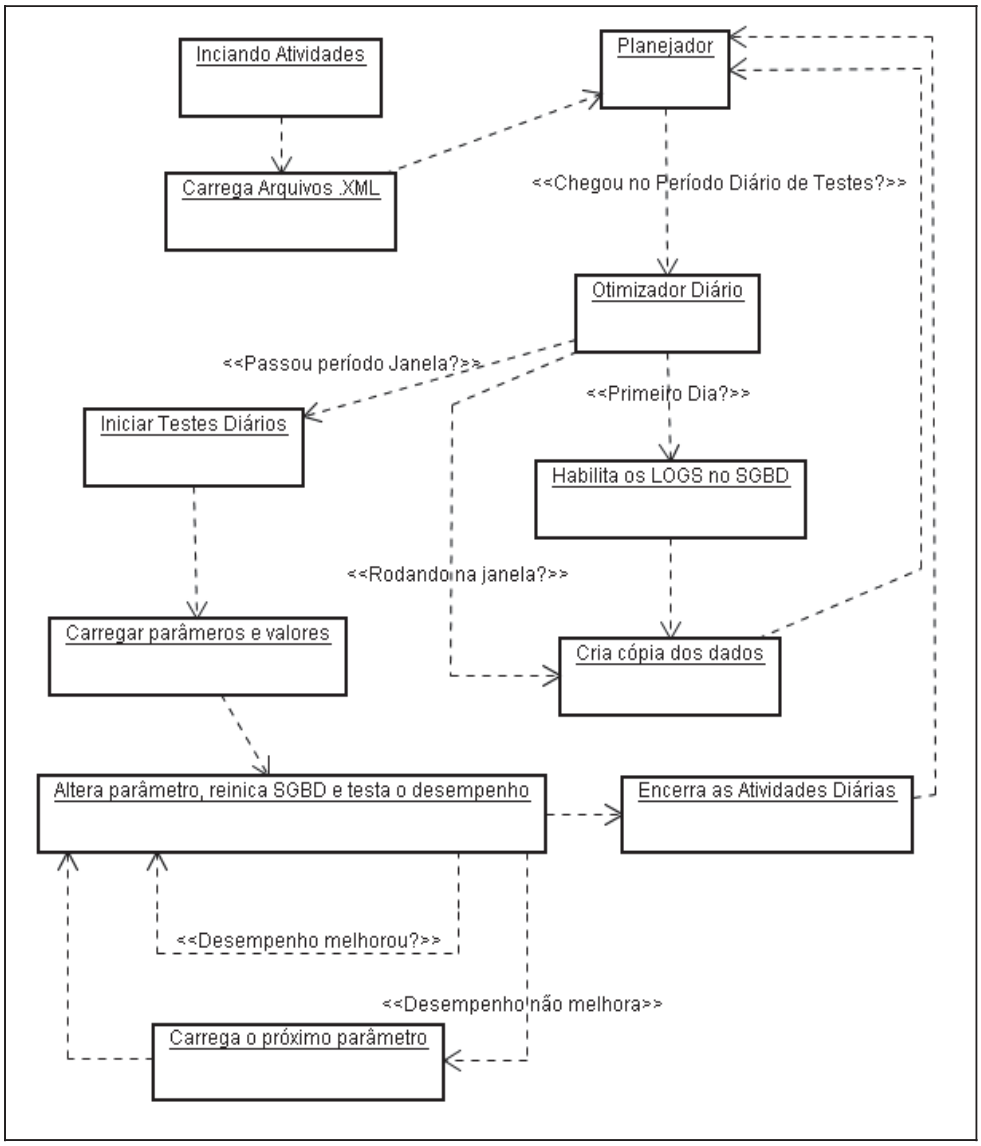

Figura 3.1: Fluxo de Execução da Ferramenta ADA.
Fonte: MACHADO e NASCIMENTO (2009).

Através da figura 3.1 é possível um melhor entendimento a respeito do funcionamento do agente. Após a inicialização das atividades, o planejador principal da aplicação, o qual é

responsável por disparar as tarefas certas no horário certo, permanece ativo e dá início a uma série de procedimentos que fazem parte dos testes necessários para se otimizar o desempenho do SGBD. O planejador é o responsável por iniciar no momento correto o otimizador, sendo que este, por sua vez, controla todos os testes e verifica os desempenhos obtidos com as diversas configurações utilizadas nos parâmetros. Após um período de janela pré-estabelecido pelo usuário, o agente está pronto para iniciar os testes. Para isto, permanece alterando os itens e verificando a melhoria obtida no desempenho, até o momento em que encerra as suas atividades diárias, seja por tempo limite atingido, seja pelo fato de que a performance ideal já foi atingida para os parâmetros trabalhados.

De acordo com os autores, ADA é um sistema autonômico capaz de, gradualmente, sintonizar os parâmetros em um SGBD PostgreSQL que possuem relação com o seu desempenho, levando em conta a carga real que está sendo submetida ao banco de dados e o ambiente computacional onde o mesmo está inserido, com a finalidade de melhorar a sua performance. A ferramenta ADA é um agente, conectado ao SGBD PostgreSQL, com o principal objetivo de realizar a correta configuração dos itens do arquivo postgresql.conf, utilizando a carga real que é executada no SGBD para testar as configurações calculadas e assim verificar se o desempenho do SGBD está realmente aumentando com as intervenções efetuadas.

3.2 Computação Autonômica em SGBDs

Elnaffar et al. (2003) apresenta um estudo onde se verificou até que ponto os SGBDs estão autonômicos. As características autonômicas foram pesquisadas, exploradas, e os resultados mostram o ponto até onde os SGBDs tomam proveito dos benefícios da computação autonômica. Um SGBD que se utilize de computação autonômica pode ser capaz de efetuar refinamentos contínuos no seu funcionamento, fazendo com que o seu desempenho seja melhorado, sua segurança aumentada, entre outros. No artigo, são examinadas as características que um SGBD deve possuir para que seja considerado autonômico, e são avaliadas as atuais situações de SGBDs comerciais como DB2, Oracle e SQL Server.

Todas as características da computação autonômica são discutidas, porém neste trabalho a ênfase foi dada para a auto-otimização.

> Auto-otimização é uma das funcionalidades mais desafiadoras a ser incluída em um SGBD. Ela permite que um SGBD execute qualquer tarefa ou serviço da maneira mais eficiente dados os parâmetros atuais, recursos disponíveis, e configurações de ambiente (ELNAFFAR et al., 2003, p. 1).

A referida pesquisa conclui que, apesar de os SGBDs comerciais possuírem alguns itens autonômicos já disponíveis, ainda há um longo caminho a ser percorrido até que se afirme que os SGBDs atuais são sistemas autonômicos.

Apesar desse cenário, a IBM tem se destacado no ramo da computação autonômica. No ano de 2002, a empresa desenvolveu o LEO (*LEarning Optmizer*), um protótipo capaz de melhorar o desempenho de consultas no SGBD DB2 em determinadas ocasiões, e sob determinadas situações (MARKL, 2003). De acordo com Markl (2003), LEO é um otimizador autonômico que observa a execução atual de consultas no SGBD DB2 e utiliza as cardinalidades atuais para autonomicamente validar e refinar as estimativas de seu modelo.

Desta maneira, otimiza-se a consulta atual, e uma futura consulta que utilize cardinalidades semelhantes às calculadas para a consulta já executada se beneficiará desta otimização. A figura 3.2 demonstra o funcionamento do LEO, mostrando como o laço de *feedback* é utilizado pra alimentar o sistema em cada volta. O laço inicia na compilação de SQL, e segue através do otimizador para o melhor plano.

Conforme pode ser observado na figura 3.2, a arquitetura do LEO é construída de acordo com o laço autonômico composto de quatro passos: monitorar, analisar, *feedback* e exploração do *feedback*.

Figura 3.2: O Laço de *Feedback* do LEO
Fonte: MARKL (2003).

Seguindo a linha de computação autonômica aplicada a SGBDs, Wiese et al (2008) propuseram a criação de um *framework* para sintonia autonômica de SGBDs orientado a melhores práticas. O maior problema enfrentado pelos pesquisadores foi encontrar uma maneira de se transmitir o conhecimento dos administradores de bancos de dados acerca de sintonia para a ferramenta desenvolvida. Para a formalização deste conhecimento do administrador do banco de dados, algumas divisões foram feitas, de modo que a ferramenta absorvesse as técnicas. As interfaces que corresponderam aos meios de se efetuar a sintonia no banco de dados foram diversas, dependendo de cada situação, e estão dispostas no quadro 3.2.

No referido estudo, os autores apresentam detalhes da implementação da solução desenvolvida, os componentes especificados e criados, e mostram através de testes conhecidos os resultados aos quais se chegou após o uso da ferramenta. Enquanto que um sistema sem a sintonia sofreu claramente um problema de baixa "*Buffer pool hit ratio*" (BPHR, a medida de quão frequente um acesso a uma página é realizado sem requerer um acesso a disco), o qual afetou o desempenho drasticamente, o sistema com a sintonia do framework teve sua BPHR aumentada pelo tamanho do *bufferpool* a cada vez que o *threshold* correspondente foi violado, o que melhorou este índice.

Interface	Descrição
WIN	Comandos do sistema operacional Windows. Por exemplo: Ler e escrever arquivos/variáveis, executar scripts, etc.
DB2_SYS	Comandos do sistema que permitem acesso e manutenção ao gerenciador do banco de dados.
DB2_CLP	Processamento de linhas de comando que permitem a execução de utilitários do banco de dados.
DB2_SQL	Instruções SQL. Por exemplo: connect, commit; comandos DML (select, update, delete); comandos DDL (create/alter/drop).
DBA	Interação com os administradores do banco de dados e da aplicação.
PE	Interação com a ferramenta DB2 Performance Expert, através de uma API.
ATE	Interface ATE para acesso aos metadados e (re)configuração do comportamento do sistema de sintonia.

Quadro 3.2: Interfaces dos componentes do sistema para os passos da sintonia
Fonte: WIESE et al., 2008

3.3 Uso de Agentes de Software para melhoria de desempenho em SGBDs

Salles e Lifschitz (2004) construíram um agente que coleta comandos SQL executados pelo SGBD e após este processo analisa quais índices seriam adequados para estes comandos e os cria automaticamente. A criação de índices torna a execução de uma aplicação de banco de dados mais rápida. Por mais rápida pode-se entender que a aplicação terá mais vazão em termos de transações que executa ou que algumas de suas transações terão menores tempos de resposta.

Os autores mostram que uma atividade comumente realizada por administradores de bancos de dados para acelerar o desempenho de consultas submetidas a um SGBD relacional é a seleção de índices sobre as tabelas presentes na base de dados. A automatização da atividade de criação de índices envolve diversas considerações, como, por exemplo, a obtenção da carga de trabalho adequada, a escolha de quais tabelas e colunas devem ser indexadas, e a determinação do momento adequado para criar ou remover índices.

Foi proposto o uso de um agente de software embutido no SGBD, o qual obtém os comandos processados pelo SGBD e decide quando criar índices adequados para estes comandos.

Devido à alta complexidade das implementações de SGBDs, soluções de auto-sintonia

ainda são muito restritas (COSTA et al., 2005). É necessário adaptar a arquitetura de um SGBD para que se tenha um comportamento automático. No trabalho apresentado por Costa et al. (2005), os autores consideram o uso de agentes de software para trabalhos de auto-sintonia e requerimentos operacionais de SGBDs.

Através da implementação de uma arquitetura no SGBD PostgreSQL, os autores mostram que, dado um conjunto de heurísticas sobre a decisão de criação ou exclusão de índices no SGBD, os agentes habilitam um comportamento automático de uma maneira bastante efetiva, integrados ao otimizador do SGBD.

Costa et al. (2005) explica que foi utilizada uma arquitetura em camadas, as quais são apresentadas abaixo:

- Sensores: Camada responsável por obter informação dos componentes do SGBD;
- Crenças: Esta camada armazena informação sobre as tabelas do SGBD, índices atuais e índices candidatos;
- Raciocínio: Esta camada avalia as crenças dos agentes e enumera possíveis cursos de ações;
- Ação: Esta camada efetivamente executa as modificações decididas pela camada de raciocínio.

Após a definição das camadas e da criação da arquitetura, a mesma foi compilada juntamente com o SGBD PostgreSQL para que ambos trabalhassem de forma integrada. Como o agente foi compilado junto ao código do SGBD, ele possui acesso a todas as funções do SGBD. Desta maneira, quando o agente decide que precisa efetuar uma ação como a criação de um índice, por exemplo, ele simplesmente acessa as interfaces da implementação do SGBD que executam a tarefa desejada (COSTA et al., 2005).

Embora seja sabido que uma sintonia fina é possível através da configuração dos parâmetros de um SGBD, alguns trabalhos também foram realizados a respeito do uso correto de índices pelo sistema. Em Morelli et al. (2009), os autores propõem uma solução que decide o momento correto para a recriação dos índices de um determinado SGBD. A solução proposta está baseada no uso de agentes para a implementação das funcionalidades da ferramenta.

A solução proposta está baseada no uso de agentes para a implementação das funcionalidades da ferramenta.

Morelli et al. (2009) esclarece que as estratégias que foram utilizadas são baseadas em heurísticas *ad-hoc*. Estas heurísticas são responsáveis pelo monitoramento das estruturas dos

índices existentes no SGBD, no que tange o nível de fragmentação destas estruturas. Sempre que estas heurísticas decidem que as estruturas dos índices estão em um nível de fragmentação não aceitável, a ferramenta se encarrega da recriação destas estruturas dos índices. Os autores esclarecem que ainda há muito desconhecimento sobre o momento ideal de criar ou remover um determinado índice e, principalmente, de quando recriá-lo.

De acordo com os autores, o benefício trazido pela existência de um determinado índice, para uma determinada consulta, é influenciado por dois fatores: a seletividade da consulta e o grau de fragmentação do índice. Logo, manter índices fragmentados pode degradar o desempenho do SGBD.

A figura 3.3 apresenta um índice que não sofreu nenhum particionamento de páginas de disco.

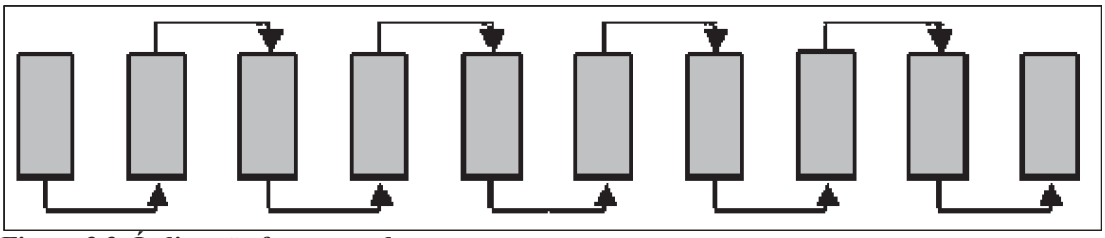

Figura 3.3: Índice não fragmentado.
Fonte: Morelli et al. (2009)

A figura 3.4 apresenta um índice fragmentado, ou seja, o mesmo índice da figura 10, após sofrer várias atualizações, o que ocasionou um eventual particionamento de páginas de disco do índice.

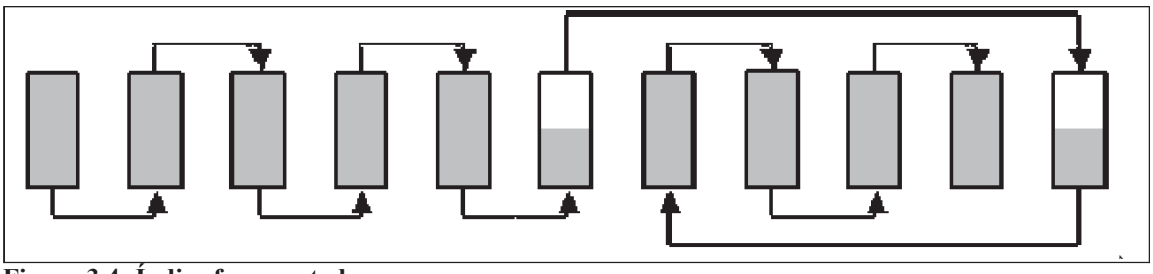

Figura 3.4: Índice fragmentado.
Fonte: Morelli et al. (2009)

Morelli et al. (2009) esclarece que as setas apresentadas nas figuras 10 e 11 revelam a

ordem na qual o índice será percorrido. Os autores salientam que deve-se perceber a ausência de múltiplos deslocamentos a posições anteriores em um índice apresentando grau de fragmentação zero.

Os autores do estudo descrevem que o contrário acontece com um índice fragmentado, o mesmo prejudica as operações de varredura, já que o artefato físico responsável pela leitura pode ser obrigado a realizar muitos deslocamentos físicos. É importante lembrar que a preocupação com a fragmentação está relacionada à varredura do índice e não aos acessos individuais as páginas em disco.

Baseados no problema apresentado, os autores desenvolveram a ferramenta utilizando o SGBD *PostgreSQL* (POSTGRESQL, 2010). A ferramenta é capaz de decidir o momento exato da recriação dos índices do SGBD, e testes foram executados para a validação da mesma.

Os testes envolveram uma única tabela com 400 mil tuplas. Nos testes procurou-se observar o comportamento dos agentes desenvolvidos, bem como a criação, destruição e recriação de índices.

> A abordagem proposta executa de forma contínua e automática, ou seja, independentemente de interações com seres humanos. Especificamente, desenvolvemos um conjunto de heurísticas que executam continuamente, detectam a existência de índices fragmentados, e, sempre que necessário, reconstroem estes índices, evitando assim os malefícios causados pela existência de índices fragmentados (MORELLI et al., 2009, p. 14).

3.4 Análise Comparativa dos Trabalhos Correlatos

Os trabalhos apresentados apresentam similaridades com esta pesquisa, porém demonstram limitações e diferenças em relação ao conteúdo desenvolvido quando comparados com a presente pesquisa.

O quadro 3.3 estabelece uma comparação entre os trabalhos relacionados aqui apresentados. Demonstra os pontos fortes de cada trabalho apresentado bem como os diferenciais da pesquisa atual em relação aos mesmos.

Trabalho	Pontos Fortes	Diferenciais desta Pesquisa
SARD (DEBNATH, LILJA E MOKBEL, 2008)	Se utiliza de um modelo matemático conciso e técnicas de estatística para encontrar os parâmetros que efetivamente são os gargalos relacionados ao desempenho de um SGBD. Após encontrar estes parâmetros, monta um ranking com os mesmos, após uma série de testes TPC-H em um SGBD PostgreSQL.	Este trabalho relacionado não encontra, de fato, os melhores valores para cada parâmetro. Apenas identifica o quão importante cada um dos parâmetros é para o desempenho do SGBD. Sendo assim, o mesmo não melhora o desempenho do SGBD, tampouco sugere ao usuário qual o melhor conjunto de parâmetros. A presente pesquisa faz isto. Utiliza-se de um modelo bayesiano e de testes de *benchmark* para, de maneira efetiva, fornecer os melhores valores para os parâmetros que afetam o desempenho do SGBD que está sendo trabalhado.
ADA (MACHADO E NASCIMENTO, 2008)	Realiza de forma empírica testes com os valores dos parâmetros do SGBD PostgreSQL que refletem em seu desempenho para encontrar os melhores valores para estes parâmetros.	Os meios utilizados para se obter a sintonia são diferentes. ADA realiza testes em um ambiente real, acoplado ao SGBD, porém não se utiliza de técnicas de mineração de dados tampouco considera de modelos probabilísticos dos regimes de operação do banco. Os testes feitos por ADA são lineares e extensivos.
SGBDs atuais: O quanto são autonômicos? (ELNAFFAR et al., 2003)	Vasculha os SGBDs atuais em busca de características autonômicas e determina, ao final, qual o nível de computação autonômica atualmente implementado nos SGBDs conhecidos.	Este trabalho é apenas um estudo para constatar que muito ainda se falta para que possa dizer que os SGBDs atuais são realmente autonômicos. Apesar disto, este estudo não apresenta qualquer implementação. A presente pesquisa propõe uma arquitetura autonômica para que se trabalhe com um dos pilares da computação autonômica: a auto-otimização.
LEO (MARKL, 2003)	Na época de publicação o estudo foi pioneiro e trouxe muitos avanços, servindo de base para trabalhos posteriores envolvendo SGBDs e computação autonômica. Mostra claramente os primeiros passos necessários para uma solução autonômica associada a um SGBD.	A presente pesquisa, apesar de também trabalhar com uma arquitetura de agentes distribuídas ao longo do laço de controle autonômico, utiliza-se de técnicas técnicas diferentes envolvendo, primeiro a mineração de dados e modelos probabilísticos para identificar o regime de de trabalho do SGBD e posteriormente, através de um modelo otimização baseado em experimentação controlada, encontrar um melhor conjunto de valores para os parâmetros de um SGBD específico (PostgreSQL). Sendo assim, apesar de o objetivo de ambas serem a melhoria do desempenho do SGBD, os meios utilizados e os locais nos quais se busca esta melhoria são completamente diferentes.
Criação de um framework para sintonia autonômica ... (WIESE et al., 2008)	A arquitetura utilizada e a metodologia que foi seguida mostraram a eficácia dos modelos adotados.	O *framework* criado formaliza uma série de boas práticas que devem ser utilizadas no momento de efetuar a sintonia de um SGBD. Apesar de diversos testes haverem sido realizados em um SGBD real, o trabalho não encontra os melhores valores para os parâmetros trabalhados. A presente pesquisa faz isto.

Trabalho	Pontos Fortes	Diferenciais desta Pesquisa
Uso de Agentes de Software para Melhoria de Desempenho em SGBDs (SALLES E LIFSCHITZ, 2004)	Através do uso de agentes, foi feita a adaptação da estrutura do SGBD para a obtenção de comportamento autonômico.	Apesar da eficácia comprovada da técnica utilizada, e dos resultados obtidos, apenas um item entre vários está sendo trabalhado para a melhoria do desempenho do SGBD: a criação de índices. Neste trabalho relacionado, outros fatores ligados ao desempenho são deixados de lado, como a sintonia de parâmetros e o aproveitamento de recursos disponíveis de hardware. Por outro lado, estes fatores são considerados na presente pesquisa.
Implementação de uma arquitetura baseada em agentes para sintonia ... (COSTA et al., 2005)	O uso de um arquitetura baseada em agentes e os trabalhos com auto-sintonia são os pontos fortes deste trabalho.	O trabalho relacionado, da mesma forma que o trabalho acima, apenas trabalha com a recriação de índices como item para se alcançar a melhoria de desempenho. A presente pesquisa difere pois encontra os melhores valores para os parâmetros que afetam o desempenho de um SGBD.
Reindexação Automática em SGBDs relacionais (MORELLI et al., 2009)	Trabalho bastante atual, onde os autores efetuam a recriação de índices de maneira automática, utilizando-se de princípios da computação autonômica.	O trabalho relacionado não cita a sintonia de SGBDs através de uma mudança fina nos parâmetros, mas apenas trata a melhora no desempenho do SGBD através da recriação de índices.

Quadro 3.3: Comparação dos trabalhos relacionados com a pesquisa atual.

Em resumo, o trabalho que está sendo apresentado aqui possui diversos pontos que não foram encontrados na literatura e nos trabalhos pesquisados. Um fator claramente visível e que não foi empregado anteriormente em qualquer estudo se trata do uso de técnicas de mineração de dados e de modelos probabilísticos para identificar o regime de operação de um SGBD.

A divisão do processo de otimização entre a identificação do regime de operação e a posterior busca dos valores ótimos dos parâmetros de configuração também é um importante diferencial do presente trabalho, não encontrando similar na literatura pesquisada.

O modelo de otimização de parâmetros é parcialmente baseado nas técnicas de ordenamento (*ranking*) de parâmetros utilizadas em alguns trabalhos relacionados, porém se diferencia daqueles utilizados nos trabalhos relacionados em dois pontos importantes: (a) a pesquisa de valores ótimos orientada por regime de operação e (b) um processo de pesquisa desses valores que utilizados mecanismos não-lineares de busca que permitiram refinamentos importantes nos valores encontrados.

Outro aspecto importante desta pesquisa é integração concreta da arquitetura de laço autonômico em uma estrutura de sistema multiagente. Isso foi possível pela utilização, desde o início do projeto, de técnicas de engenharia de software baseada em agentes

(HENDERSON-SELLERS e GIORGINI, 2005) para a construção da ferramenta de sintonia.

Conceitos como autonomia e pro-atividade são parte inerente da programação e modelagem de sistemas baseados em agentes (WOOLDRIDGE, 2002)(BORDINI et al., 2005). Tais conceitos também são fundamentais para sistemas de auto-sintonia. A programação de processos de decisão baseados em mecanismos de raciocínio prático também é uma parte inerente da programação baseada em agentes (WOOLDRIDGE, 2002). Tais procedimentos também são fundamentais para a construção de sistemas realmente autônomos. Essas afinidades serviram de base para a definição da arquitetura de laço autonômico baseada em agentes, proposta nesse trabalho para realizar a otimização de SGBDs.

4 MODELO DE IDENTIFICAÇÃO DE REGIME DE OPERAÇÃO

Este capítulo descreve o modelo de identificação do regime de operação do SGBD, modelo este que foi criado para ser utilizado pela ferramenta desenvolvida. O modelo é detalhado em suas fases de definição e de criação.

4.1 Definição do Modelo de Identificação do Regime de Operação

Uma tarefa importante desempenhada pela ferramenta desenvolvida consiste na correta classificação do regime de operação desempenhado pelo SGBD que está sendo utilizado. Para tanto, trabalhou-se com a definição de um modelo, próprio para esta tarefa de classificação.

4.1.1 Definição dos Regimes de Operação Utilizados no Modelo

Para que a caracterização de carga submetida a um SGBD possa ser corretamente classificada, o modelo desenvolvido define categorias de uso comuns aos SGBDs. Estas categorias foram denominadas de regimes de operação, e foram definidas para representar um possível padrão de utilização do SGBD. Os regimes de operação trabalhados no modelo são os seguintes:

a) Padrão OLTP: O regime de operação OLTP (O*nLine Transaction Processing*) refere-se ao uso de um SGBD para o processamento *on-line* de transações, através de uma rede. As aplicações OLTP permitem atualizações constantes de dados (ou seja, as informações são modificadas diariamente ou periodicamente). Isto consiste de instruções de seleção, inserção, atualização e deleção sendo executadas periodicamente por um SGBD. Este regime de operação também refere-se ao SGBD utilizado para atender às demandas de aplicações comerciais ou industriais;

b) Padrão Web: Este regime de operação refere-se ao SGBD utilizado em conjunto com um sítio na Internet, portal de internet, intranet, ou qualquer outra aplicação que execute

através de um navegador web, e cuja finalidade seja informativa. Como a maior parte das transações executadas a partir de um sítio da internet são apenas consultas a bancos de dados, esta categoria foi criada para cobrir este padrão de comportamento do SGBD;

c) Padrão Produção: Nesta categoria são enquadrados os SGBDs que precisam atender às instruções de inserção, seleção, deleção e atualização de uma maneira uniforme, geralmente em larga escala, executando funções de atualização ou processamento de grandes conjuntos de dados.

Outros regimes de operação podem ser criados e adicionados ao modelo, para que uma variedade bastante grande de padrões de usos de SGBDs possa ser coberta pela aplicação. No entanto, os regimes de operação já definidos no modelo cobrem a grande maioria de usos definidos para os SGBDs encontrados na atualidade.

Para cada uma dos regimes de operação definidos no modelo, há um conjunto de transações correspondente, o qual caracteriza a categoria em questão. Estes conjuntos de transações são sugeridos pela ferramenta pgbench, a qual foi utilizada nesta pesquisa.

De acordo com PGBench (2010), pgbench é um programa simples para execução de testes de *benchmark* no SGBD PostgreSQL. Ele roda a mesma sequência de comandos SQL por diversas vezes, com a possibilidade de simulação de vários clientes conectados ao mesmo tempo.

A figura 4.1 demonstra o conjunto de comandos para a transação que é executada para o regime de operação OLTP.

```
1. BEGIN;
2. UPDATE pgbench_accounts SET abalance = abalance + :delta WHERE aid = :aid;
3. SELECT abalance FROM pgbench_accounts WHERE aid = :aid;
4. INSERT INTO pgbench_history (tid, bid, aid, delta, mtime) VALUES (:tid, :bid,
   :aid, :delta, CURRENT_TIMESTAMP);
5. END;
```

Figura 4.1: Conjunto de comandos para o tipo de transação OLTP.

A figura 4.2 demonstra o conjunto de comandos para a transação que é executada para o regime de operação web.

```
1. BEGIN;
2. SELECT abalance FROM pgbench_accounts WHERE aid = :aid;
3. END;
```

Figura 4.2: Conjunto de comandos para o tipo de transação Padrão Web.

A figura 4.3 demonstra o conjunto de comandos para a transação que é executada para o regime de operação produção.

```
1. BEGIN;
2. UPDATE pgbench_accounts SET abalance = abalance + :delta WHERE aid = :aid;
3. SELECT abalance FROM pgbench_accounts WHERE aid = :aid;
4. UPDATE pgbench_tellers SET tbalance = tbalance + :delta WHERE tid = :tid;
5. UPDATE pgbench_branches SET bbalance = bbalance + :delta WHERE bid = :bid;
6. INSERT INTO pgbench_history (tid, bid, aid, delta, mtime) VALUES (:tid, :bid,
   :aid, :delta, CURRENT_TIMESTAMP);
7. END;
```

Figura 4.3: Conjunto de comandos para o tipo de transação Padrão Produção.

Para se atingir os resultados que dele se esperam, o modelo precisa estar treinado para que, no momento em que recebe um conjunto de transações realizadas em um SGBD, consiga efetuar a classificação deste SGBD em um dos regimes de operação definidos acima.

4.1.2 Seleção do Algoritmo Utilizado para Treinamento do modelo

Como parte da definição do modelo, decidiu-se a utilização de um classificador bayesiano simples (*naive-bayes)*. Esta decisão foi fundamentada no fato de que os outros algoritmos testados não demonstraram uma taxa de acurácia tão alta quanto as obtidas pelo classificador bayesiano simples.

De acordo com Markov e Larose (2007), as medidas mais corretas para se avaliar o

desempenho de um classificador são a acurácia e a taxa geral de erros. A acurácia é definida como o número verdadeiros positivos adicionado ao número de verdadeiros negativos, sendo o resultado dividido pelo número total de registros. A taxa geral de erros é definida pelo número de falsos positivos, adicionado ao número de falsos negativos, sendo o resultado dividido pelo número total de registros. Como pode-se observar, ambas são complementares.

O quadro 4.1 demonstra as taxas de acurácia e as taxas de erros encontradas com os algoritmos testados, fundamentando assim a escolha pelo classificador bayesiano simples.

	Naive-Bayes	KNN	Redes Neurais
Taxa de acurácia	85,33 %	74,19%	80,12%
Taxa de erros	14,67 %	25,81%	19,88%

Quadro 4.1: Taxas de Acurácia e Taxas de Erros encontradas com os algoritmos testados

As taxas foram encontradas através da aplicação dos algoritmos escolhidos no modelo criado na ferramenta RapidMiner, o qual será explicado na seção 4.2. Como exemplo deste processo, a figura 4.4 demonstra a matriz de confusão gerada pela ferramenta RapidMiner para o algoritmo KNN.

KNN

Matriz de confusão

	true Padrão OLTP	true Padrão Web	true Padrão Produção	class precision
pred. Padrão OLTP	50	25	3	64,10%
pred. Padrão Web	7	34	2	79,07%
pred. Padrão Produção	4	1	37	88,10%
class recall	81,97%	56,67%	88,10%	

Acuracidade	74,19%	+/- 9,22%	(mikro: 74,23%)
Erros de classificação	25,81%	+/- 9,22%	(mikro: 25,77%)
Erros relativos	25,81%	+/- 9,22%	(mikro: 25,77% +/- 43.74%)

Figura 4.4: Matriz de Confusão para o algoritmo KNN testado

Como pode ser observado, a decisão pelo algoritmo classificador bayesiano simples permitiu, no cenário utilizado, uma melhor taxa de acurácia para a classificação dos regimes

de operação.

4.1.3 *Overfitting* do Modelo

A fim de evitar o fato conhecido como *overfitting* do modelo de identificação de regime de operação, foi acrescido algum ruído aos valores gerados no arquivo .CSV. De acordo com Tan, Steinbach e Kumar (2009), um dos motivos para a ocorrência de *overfitting* ocorre quando um modelo de classificação se adapta muito bem aos dados de treinamento, porém tem dificuldades em classificar com precisão registros que nunca viu anteriormente. Ainda de acordo com Tan, Steinbach e Kumar (2009), isto é importante porque um modelo que seja apropriado aos dados de treinamento pode muito bem ter um erro de generalização mais pobre do que um modelo com um alto grau de erro de trinamento. Este foi o problema encontrado neste momento dos trabalhos com o modelo utilizado. Encontrou-se problemas ao utilizar o arquivo .CSV original, sem qualquer alteração, pois os algoritmos utilizados para a classificação chegavam muito próximos a 100% de acurácia, ou seja, quase 0% de erros. Um classificador com estas taxas, baseando-se apenas em valores perfeitos utilizados para o seu treinamento, encontra dificuldades no momento de classificar outras entradas que não aquelas utilizadas em seu treino.

Para gerar-se o ruído no arquivo .CSV, sorteou-se um percentual de aleatoriedade entre -20% e 20% para cada um dos valores encontrados para as instruções de *select, insert, update* e *delete* utilizadas. Com os valores alterados, as novas taxas de acurácia e de erros demonstradas anteriormente foram encontradas, e isto permitiu que o modelo treinado pudesse lidar de forma mais acertada com os valores ainda não conhecidos, os quais foram utilizados para os testes e avaliação de resultados do protótipo desenvolvido.

4.2 Criação do Modelo de Identificação de Regime de Operação

Após a definição do modelo a ser construído, partiu-se para a fase de criação do mesmo. Para a criação do modelo de identificação do regime de operação, foi utilizada a ferramenta RapidMiner.

A ferramenta RapidMiner foi selecionada por ser a melhor entre as mais importantes ferramentas de mineração de dados de código aberto em termos de tecnologia e aplicabilidade (Rapid-I, 2010). Além disso, a ferramenta Rapidminer oferece suporte ao uso do algoritmo Naive Bayes, bem como o uso dos principais algoritmos de classificação conhecidos.

A figura 4.5 ilustra o processo criado para o modelo de identificação de regime de operação na ferramenta RapidMiner.

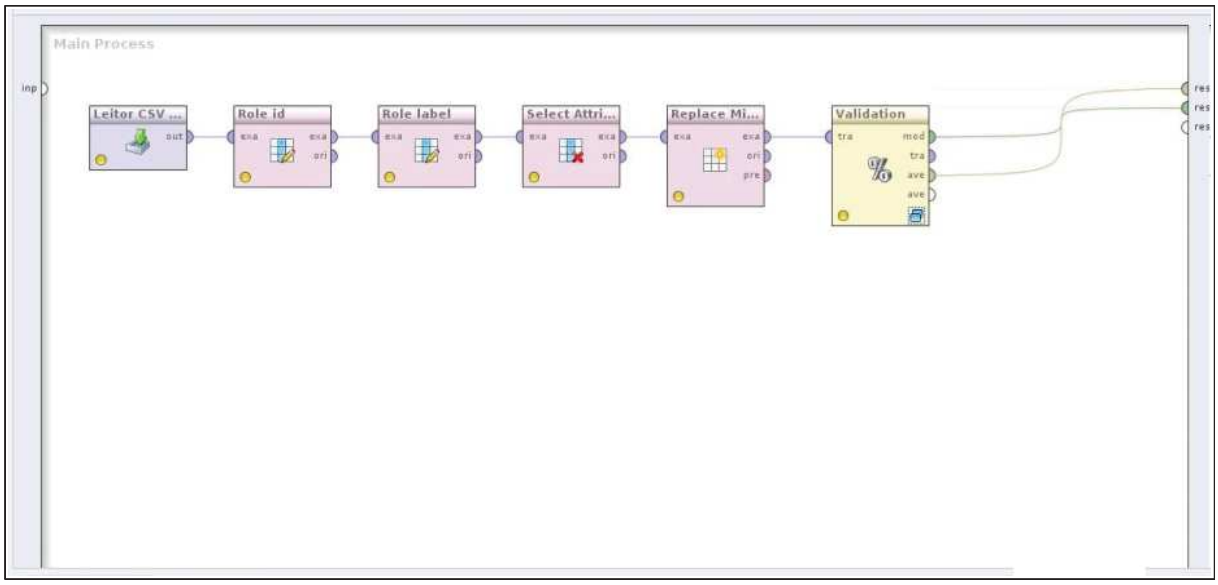

Figura 4.5: Processo para o modelo de identificação de regime de operação na ferramenta RapidMiner

Como pode ser observado na figura 4.5, o processo correspondente ao modelo de identificação de regime de operação é composto por alguns operadores, os quais são ligados entre si produzindo as saídas esperadas.

O primeiro operador instanciado é o operador de leitura do arquivo CSV contendo os dados referentes às transações executadas no SGBD. Um exemplo de arquivo CSV é demonstrado na figura 4.6.

Tempo	Selects	Inserts	Updates	Deletes	Classe
143	31.25	30,93	30,93	7,22	Padrão OLTP
144	37,20	28,99	28,99	4,83	Padrão OLTP
145	34,58	28,04	35,05	2,34	Padrão OLTP
146	31,82	33,33	30,30	4,55	Padrão OLTP
147	32,80	35,45	31,75	0,00	Padrão OLTP
148	35,50	34,50	30,00	0,00	Padrão OLTP
149	30,15	30,15	33,17	6,53	Padrão OLTP
150	35,78	34,80	29,41	0,00	Padrão OLTP
151	35,82	29,85	34,33	0,00	Padrão OLTP
152	34,48	29,56	35,96	0,00	Padrão OLTP
153	38,94	32,21	28,85	0,00	Padrão OLTP
154	32,79	34,43	32,79	0,00	Padrão OLTP
155	31,75	36,51	31,75	0,00	Padrão OLTP
156	32,09	35,29	32,09	0,53	Padrão OLTP
157	31,75	35,45	32,80	0,00	Padrão OLTP
158	33,33	30,77	33,33	2,56	Padrão OLTP
159	33,33	33,33	33,33	0,00	Padrão OLTP
160	35,43	31,84	26,91	5,83	Padrão OLTP
161	32,79	32,79	32,79	1,64	Padrão OLTP
162	30,46	39,09	30,46	0,00	Padrão OLTP
163	31,41	37,17	31,41	0,00	Padrão OLTP
164	35,83	32,09	32,09	0,00	Padrão OLTP
165	29,56	29,56	40,89	0,00	Padrão OLTP
166	34,16	29,70	31,19	4,95	Padrão OLTP

Figura 4.6: Arquivo de entrada do processo

No operador denominado “Leitor CSV” estão mapeados o arquivo que será utilizado, bem como as colunas existentes neste arquivo CSV gerado. O arquivo CSV contém o instante de tempo coletado para as transações informadas (este instante de tempo é um item lógico e definido na aplicação *AtTuneDB*), o percentual de *selects*, *inserts*, *updates* e *deletes* no instante de tempo informado, e a classe correspondente a estas transações.

Os operadores “Role id” e “Role label” aplicam papéis às colunas. O operador “Role id” identifica a coluna de tempo como sendo o identificador de cada uma das linhas. O operador “Role label” serve, por exemplo, para se definir que a coluna “Classe” do arquivo CSV acima contém o nome da classe a ser utilizada pelo classificador.

O operador “Select Attributes” permite que se selecione quais são os atributos (ou colunas) que serão utilizados no processo. No caso do exemplo acima todos os atributos foram selecionados.

O operador “Replace Missing Values” permite que se defina um valor padrão caso nenhum esteja informado no arquivo CSV. Na criação do modelo o valor 0 (zero) é definido quando algum valor está faltando, no entanto isto não ocorreu pois todos os valores estão presentes nos testes realizados. Este operador apenas foi instanciado e faz parte do modelo para que se evite falhas decorrentes de falta de valores quando a ferramenta *AtTuneDB* estiver sendo executada.

O operador “Validation” é responsável por, efetivamente, fazer o treinamento do

classificador. Expandindo-se a caixa do operador “Validation” tem-se o subprocesso demonstrado na figura 4.7.

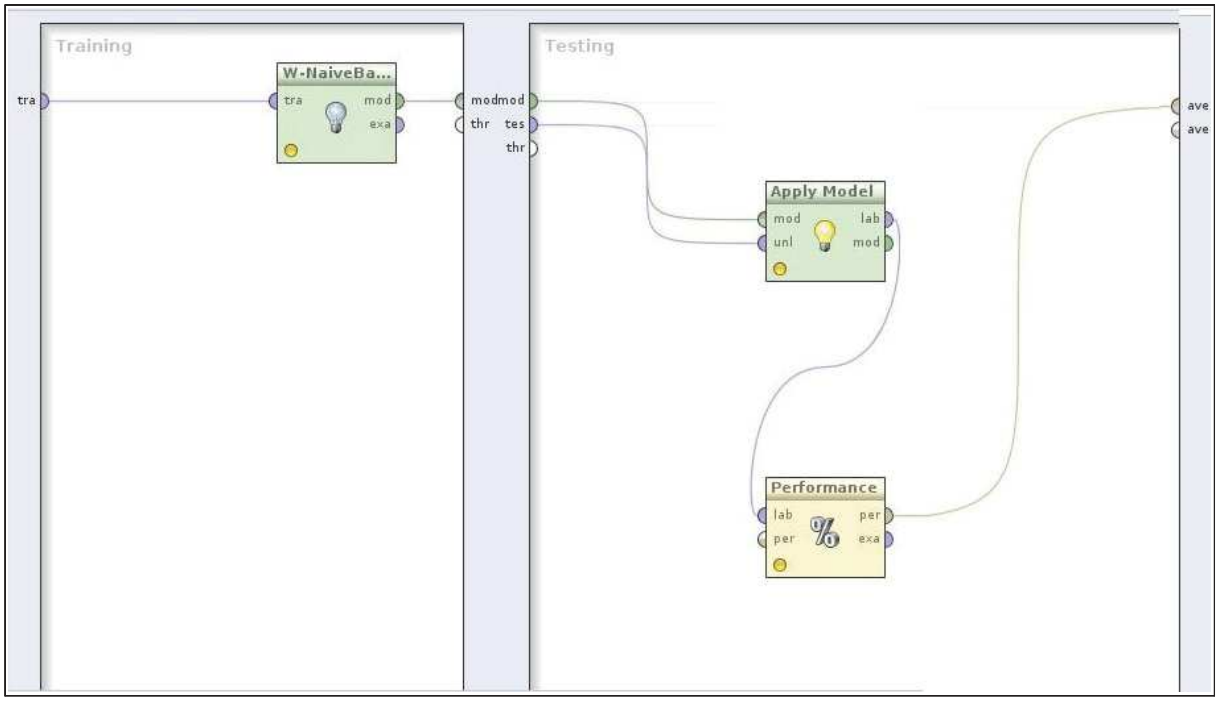

Figura 4.7: Treinamento e teste do classificador do modelo de identificação de regime de operação

O processo de validação é dividido em duas partes: treinamento e teste do classificador do modelo. Foram utilizados os percentuais de 70% para treinamento do modelo, e de 30% para testes no mesmo. No operador “Performance” foi definido que as métricas a serem utilizadas como validadoras do desempenho do classificador são as medidas de acurácia e de taxa de erros.

Como pode-se observar na caixa “Training” da figura 4.7, foi utilizado o algoritmo a implementação WEKA do classificador bayesiano simples para o treinamento (*W-NaiveBayes*). Na caixa “Testing” da figura 4.7 pode-se verificar a aplicação do modelo, bem como o operador contendo os testes de performance realizados sobre o classificador.

Assim, como saída do processo de validação, tem-se o classificador do modelo treinado com os dados recebidos, e também os resultados da aplicação do classificador. Isto permite que se extraia o classificador para fora da ferramenta RapidMiner, bem como se tenha os resultados da aplicação do mesmo na forma de uma matriz de confusão.

Este processo de obtenção dos resultados pode ser observado na figura 4.5, onde pode-se verificar que as duas saídas descritas acima são ligadas às portas de resultado do processo que representa o modelo de identificação do regime de operação.

Sendo assim, após o momento de treinamento, a ferramenta permite que se extraia o modelo que foi aprendido pelo classificador, exportando isto para um arquivo de extensão .MOD. Este arquivo pode ser explorado a partir da linguagem Java através da API apropriada, a qual é fornecida na distribuição do RapidMiner, de modo que esta será a maneira utilizada para que o agente otimizador do sistema obtenha o seu aprendizado. Utilizar o arquivo .MOD a partir da ferramenta implica no processo de aplicação do modelo sobre dados de entrada coletados do SGBD. Este processo é detalhado no capítulo 6, o qual demonstra o funcionamento da ferramenta desenvolvida.

5 MODELO DE OTIMIZAÇÃO

Este capítulo descreve o modelo de otimização que foi desenvolvido para as operações efetuadas pela ferramenta *AtTuneDB*.

5.1 Definição do Modelo

Como definição do modelo de otimização utilizado pela ferramenta desenvolvida, tem-se uma tabela com os diferentes conjuntos de parâmetros do SGBD que têm relação com o seu desempenho. Esta tabela estabelece a ordenação (*ranking*) dos parâmetros e seus valores, de acordo com uma determinada métrica de desempenho.

Este modelo caracteriza-se por uma tabela composta de linhas e colunas, onde cada linha representa um diferente conjunto de parâmetros do SGBD e cada coluna representa os possíveis valores para um parâmetro, o qual tem influência direta no resultado do desempenho do SGBD.

A primeira coluna da tabela contém o regime de operação que está sendo testado.

As colunas seguintes guardam os cruzamentos das diferentes faixas de valores definidos para os parâmetros que serão utilizados.

De acordo com as pesquisas descritas na fundamentação teórica, foram selecionados os seguintes parâmetros para otimização:

a) c*heckpoint_segments*

b) *checkpoint_timeout*

c) *shared_buffers*

d) *effective_cache_size*

e) *wal_buffers*

f) *commit_delay*

A última coluna da tabela do modelo criado refere-se aos resultados apresentados pelo SGBD quando submetido aos testes cujos resultados são apresentados nesta tabela. Este indicador foi medido através da métrica Transações Por Segundo (TPS), a qual é a norteadora,

neste trabalho, para indicações do desempenho do SGBD.

O indicador TPS mede a quantidade de transações que o SGBD foi capaz de processar em um segundo, quando utilizando o conjunto de parâmetros que está sendo trabalhado. Este indicador norteia a aplicação no momento de encontrar o resultado que deve ser fornecido ao usuário como sugestão, visto que é esta informação a que dita o melhor conjunto de parâmetros de configuração do SGBD para o conjunto de cargas que está sendo submetido aos testes.

O quadro 5.1 exemplifica o modelo definido para otimização.

Regime de Operação	Parâmetros	Resultado (TPS)
Padrão Web	shared_buffers = 28kb, wal_buffers=2mb, ...	10.5345
Padrão Web	shared_buffers = 14kb, wal_buffers=2mb, ...	9.3287
Padrão Web	shared_buffers = 28kb, wal_buffers=4mb, ...	12.9282
Padrão Web	shared_buffers = 14kb, wal_buffers=4mb, ...	11.2828
…	...	...

Quadro 5.1: Tabela de categorias, parâmetros e resultados de desempenho coletados

5.2 Criação do Modelo de Otimização

Para a criação do modelo de otimização definido na seção 5.1, foi desenvolvida uma ferramenta de apoio, a qual é responsável pela realização dos testes de *benchmark* que abastecem o o modelo.

A ferramenta foi desenvolvida na linguagem Java 1.6, utilizando-se a IDE Eclipse versão 3.5.2 (GALILEO).

5.2.1 Populando o SGBD

Antes da execução dos testes para abastecer o modelo, o SGBD utilizado foi populado. Foram criadas algumas tabelas para a manipulação dos dados, de acordo com os comandos

DDL definidos na figura 5.1.

```
CREATE TABLE pgbench_tellers
(
  tid integer NOT NULL,
  bid integer,
  tbalance integer,
  filler character(84),
  CONSTRAINT pgbench_tellers_pkey PRIMARY KEY (tid)
)
WITH (
  FILLFACTOR=100,
  OIDS=FALSE
);

CREATE TABLE pgbench_branches
(
  bid integer NOT NULL,
  bbalance integer,
  filler character(88),
  CONSTRAINT pgbench_branches_pkey PRIMARY KEY (bid)
)
WITH (
  FILLFACTOR=100,
  OIDS=FALSE
);

CREATE TABLE pgbench_accounts
(
  aid integer NOT NULL,
  bid integer,
  abalance integer,
  filler character(84),
  CONSTRAINT pgbench_accounts_pkey PRIMARY KEY (aid)
)
WITH (
  FILLFACTOR=100,
  OIDS=FALSE
);

CREATE TABLE pgbench_history
(
  tid integer,
  bid integer,
  aid integer,
  delta integer,
  mtime timestamp without time zone,
  filler character(22)
)
WITH (
  OIDS=FALSE
);
```

Figura 5.1: Comandos DDL para criação do esquema do banco de dados.

Para que se tenha uma aproximação de um SGBD real, as tabelas foram populadas com uma grande quantidade de dados, sendo que na tabela pgbench_accounts foram inseridos cem milhões de registros, de forma que o banco tenha bastantes dados e desta maneira os resultados do *benchmark* sejam confiáveis.

5.2.2 Divisão do Espaço de Pesquisa

O processo de ordenação dos valores dos parâmetros de configuração é essencialmente um processo de busca por máximos em um determinado espaço de pesquisa formado pelos valores possíveis de configurações. Dada a granulação de valores possíveis para cada parâmetro (a cardinalidade do parâmetro), o tamanho de todo os espaço de configuração, definido simplesmente pela multiplicação das cardinalidades dos parâmetros, é excessivamente grande para ser pesquisado pelo método de experimentação controlado.

Assim foi utilizado inicialmente um método baseado na divisão linear deste espaço em um subconjunto de valores igualmente espaçados para os parâmetros. O método pode ser aplicado recursivamente para identificar e refinar máximos previamente encontrados, se aproximando assim de métodos capazes de trabalhar com não-linearidades no espaço de pesquisa. O método foi executado em caráter semi-automático, podendo, entretanto, ser facilmente automatizado.

Para definição dos valores a serem utilizados por cada um dos parâmetros, foram utilizados os valores máximos e os valores mínimos de cada parâmetro. Estes valores máximo e mínimo correspondem aos valores de ponta do parâmetro em questão. Além dos valores de ponta, também foram utilizados valores intermediários, os quais foram divididos uniformemente entre as pontas, de maneira linear.

Por exemplo, para o parâmetro *effective_cache_size*, foram encontrados como valores das pontas os valores 64mb e 256mb (valores informados pela documentação do SGBD). Os valores intermediários são, dessa forma, 128mb e 192mb, no caso de se desejar que sejam 2 valores intermediários diferentes. Assim, ficam definidos 4 valores para o referido parâmetro (dois valores de ponta mais dois valores intermediários). A ferramenta desenvolvida está pronta para ser executada com mais configurações de diferentes valores, caso necessário.

Abaixo estão os valores definidos para os testes com cada um dos parâmetros:

a) *checkpoint_segments*: 1, 3, 5, 7

b) *checkpoint_timeout*: 2min, 4min, 6min, 8min

c) *shared_buffers:* 128kB, 14MB, 28MB

d) *effective_cache_size*: 64MB, 128MB, 192MB, 256MB

e) *wal_buffers*: 32kB, 64kB, 96kB, 128kB

f) *commit_delay*: 0, 4000, 8000

5.2.3 Execução da Ferramenta para Testes de *Benchmark*

Após a criação das tabelas e inserção dos dados na mesma, a ferramenta foi executada, e os resultados do *benchmark* coletados. Os testes foram realizados em um ambiente Linux Ubuntu 10.04, em um microcomputador com processador Intel Core2Duo, clock de 1,66GhZ, 2gb de memória RAM, com um HD Sata de 160gb. Tomou-se o cuidado de se isolar a execução da ferramenta para que nenhuma outra aplicação interferisse no desempenho medido.

A figura 5.2 ilustra parte dos resultados exibidos em console pela execução da ferramenta de apoio desenvolvida.

```
postgres@akbar:/home/leonardo$ java -jar benchmarkpg.jar
[AtTuneDB] ------------------------------------------------------
[AtTuneDB] 1 - Iniciando os testes com transação de opção <OLTP Padrão>
[AtTuneDB] ------------------------------------------------------
-------------------------------------------------------------------------------------------
Iniciando a execução do teste 1 de 6912
[AtTuneDB] Iniciando os testes com os seguintes valores:
{checkpoint_segments=1,   wal_buffers=32kB,   checkpoint_timeout=2min,   shared_buffers=128kB,
commit_delay=0, effective_cache_size=64MB}
[AtTuneDB] para o conjunto de transações de tipo <>
 * Stopping PostgreSQL 8.4 database server
   ...done.
[AtTuneDB] Gerado arquivo postgresql.conf com as novas configurações
 * Starting PostgreSQL 8.4 database server
   ...done.
[AtTuneDB] Iniciando testes com o pgbench
transaction type: TPC-B (sort of)
scaling factor: 1000
query mode: simple
number of clients: 1
duration: 10 s
number of transactions actually processed: 128
tps = 12.791946 (including connections establishing)
tps = 12.930570 (excluding connections establishing)
-------------------------------------------------------------------------------------------
Iniciando a execução do teste 2 de 6912
[AtTuneDB] Iniciando os testes com os seguintes valores:
{checkpoint_segments=1,   wal_buffers=32kB,   checkpoint_timeout=2min,   shared_buffers=128kB,
commit_delay=4000, effective_cache_size=64MB}
[AtTuneDB] para o conjunto de transações de tipo <>
 * Stopping PostgreSQL 8.4 database server
   ...done.
[AtTuneDB] Gerado arquivo postgresql.conf com as novas configurações
 * Starting PostgreSQL 8.4 database server
   ...done.
[AtTuneDB] Iniciando testes com o pgbench
transaction type: TPC-B (sort of)
scaling factor: 1000
query mode: simple
number of clients: 1
duration: 10 s
number of transactions actually processed: 128
tps = 12.622881 (including connections establishing)
tps = 12.628840 (excluding connections establishing)
```

```
-------------------------------------------------------------------------------------
Iniciando a execução do teste 3 de 6912
[AtTuneDB] Iniciando os testes com os seguintes valores:
{checkpoint_segments=1,   wal_buffers=32kB,   checkpoint_timeout=2min,   shared_buffers=128kB,
commit_delay=8000, effective_cache_size=64MB}
[AtTuneDB] para o conjunto de transações de tipo <>
 * Stopping PostgreSQL 8.4 database server
   ...done.
[AtTuneDB] Gerado arquivo postgresql.conf com as novas configurações
 * Starting PostgreSQL 8.4 database server
   ...done.
[AtTuneDB] Iniciando testes com o pgbench
transaction type: TPC-B (sort of)
scaling factor: 1000
query mode: simple
number of clients: 1
duration: 10 s
number of transactions actually processed: 135
tps = 13.457768 (including connections establishing)
tps = 13.464095 (excluding connections establishing)
-------------------------------------------------------------------------------------
```

Figura 5.2: Execução da ferramenta auxiliar para testes de *benchmark*.

Como pode ser observado na figura 5.2, os resultados do *benchmark* em TPS foram exibidos na console para cada combinação diferente de transação e de conjunto de parâmetros.

Os valores de parâmetros definidos previamente foram combinados um a um entre os diferentes parâmetros, o que gerou um total de 2304 testes. Cada um destes testes foi executado para os 3 tipos diferentes de conjuntos de transações (OLTP, Web e Produção), o que resultou em um total de 6912 testes. Para cada um dos testes, foi alocado o tempo de 20 segundos, para que a ferramenta pgbench não realizasse apenas 1 teste com a transação em questão, mas sim permanecesse repetindo o teste durante 20 segundos, e retornasse a média de transações por segundo encontrada no tempo executado.

5.2.4 Tabulação dos Dados Coletados

Após a finalização dos testes de *benchmark*, os resultados exibidos pela ferramenta na console foram coletados e tabulados de acordo com o quadro 5.1 definido na seção 5.1. A mesma ferramenta responsável por gerar os testes de *benchmark* também foi responsável por fazer um *parsing* do arquivo texto gerado e criar um outro arquivo contendo a tabela com os resultados em formato CSV (texto separado por vírgulas).

A tabela que contém os resultados dos testes de *benchmark* tabulados representa o

modelo de otimização criado para ser utilizado pela ferramenta *AtTuneDB*.

A figura 5.3 exemplifica dados tabulados para o regime de operação OLTP Padrão.

Resultados de Testes de Benchmark

Categoria	checkpoint_segments	checkpoint_timeout	shared_buffers	effective_cache_size	wal_buffers	commit_delay	Transações por Segundo (TPS)
OLTP Padrão	1	2min	128kB	64MB	32kB	0	12.791946
OLTP Padrão	1	2min	128kB	64MB	32kB	4000	12.622881
OLTP Padrão	1	2min	128kB	64MB	32kB	8000	13.457768
OLTP Padrão	1	2min	128kB	64MB	64kB	0	11.176484
OLTP Padrão	1	2min	128kB	64MB	64kB	4000	13.641937
OLTP Padrão	1	2min	128kB	64MB	64kB	8000	13.818333
OLTP Padrão	1	2min	128kB	64MB	96kB	0	13.459563
OLTP Padrão	1	2min	128kB	64MB	96kB	4000	13.453791
OLTP Padrão	1	2min	128kB	64MB	96kB	8000	13.146195
OLTP Padrão	1	2min	128kB	64MB	128kB	0	13.083095
OLTP Padrão	1	2min	128kB	64MB	128kB	4000	11.990714
OLTP Padrão	1	2min	128kB	64MB	128kB	8000	13.576645
OLTP Padrão	1	2min	128kB	128MB	32kB	0	11.932034
OLTP Padrão	1	2min	128kB	128MB	32kB	4000	13.349209
OLTP Padrão	1	2min	128kB	128MB	32kB	8000	11.818385
OLTP Padrão	1	2min	128kB	128MB	64kB	0	12.649174
OLTP Padrão	1	2min	128kB	128MB	64kB	4000	12.865298
OLTP Padrão	1	2min	128kB	128MB	64kB	8000	14.028930
OLTP Padrão	1	2min	128kB	128MB	96kB	0	13.100791
OLTP Padrão	1	2min	128kB	128MB	96kB	4000	14.087006
OLTP Padrão	1	2min	128kB	128MB	96kB	8000	13.351354
OLTP Padrão	1	2min	128kB	128MB	128kB	0	13.839075
OLTP Padrão	1	2min	128kB	128MB	128kB	4000	13.985503
OLTP Padrão	1	2min	128kB	128MB	128kB	8000	13.877707
OLTP Padrão	1	2min	128kB	192MB	32kB	0	13.650871
OLTP Padrão	1	2min	128kB	192MB	32kB	4000	13.465210
OLTP Padrão	1	2min	128kB	192MB	32kB	8000	11.503489
OLTP Padrão	1	2min	128kB	192MB	64kB	0	13.061689
OLTP Padrão	1	2min	128kB	192MB	64kB	4000	13.118319
OLTP Padrão	1	2min	128kB	192MB	64kB	8000	13.175551
OLTP Padrão	1	2min	128kB	192MB	96kB	0	12.990336
OLTP Padrão	1	2min	128kB	192MB	96kB	4000	12.454282
OLTP Padrão	1	2min	128kB	192MB	96kB	8000	13.431197
OLTP Padrão	1	2min	128kB	192MB	128kB	0	13.189886
OLTP Padrão	1	2min	128kB	192MB	128kB	4000	13.758415
OLTP Padrão	1	2min	128kB	192MB	128kB	8000	13.149880
OLTP Padrão	1	2min	128kB	256MB	32kB	0	14.239892
OLTP Padrão	1	2min	128kB	256MB	32kB	4000	13.956804
OLTP Padrão	1	2min	128kB	256MB	32kB	8000	12.385732
OLTP Padrão	1	2min	128kB	256MB	64kB	0	13.771874
OLTP Padrão	1	2min	128kB	256MB	64kB	4000	13.042002

Figura 5.3: Resultados do *benchmark* tabulados.

5.2.5 Refinamento da Ordenação através da Investigação de Máximos Iniciais

A ferramenta *AtTuneDB* desenvolvida baseia-se nos resultados dos testes de *benchmark* para fazer sugestões ao usuário sobre melhores conjuntos de valores para os parâmetros do SGBD que foram testados. Para tanto, a ferramenta utiliza a última coluna demonstrada no modelo na figura 5.3, a quantidade de transações por segundo.

A quantidade de transações por segundo foi medida, conforme descrito na seção 5.2.3, para diferentes conjuntos de valores para os parâmetros que interferem no desempenho do SGBD.

Conforme descrito na seção 5.1, ao todo foram trabalhados seis parâmetros que podem interferir no desempenho do SGBD. Variando-se apenas os valores utilizados para o parâmetro effective_cache_size, por exemplo, e fixando-se os valores utilizados para os outros cinco parâmetros, temos como representação do desempenho em TPS encontrado para o SGBD trabalhado no gráfico demonstrado na figura 5.4.

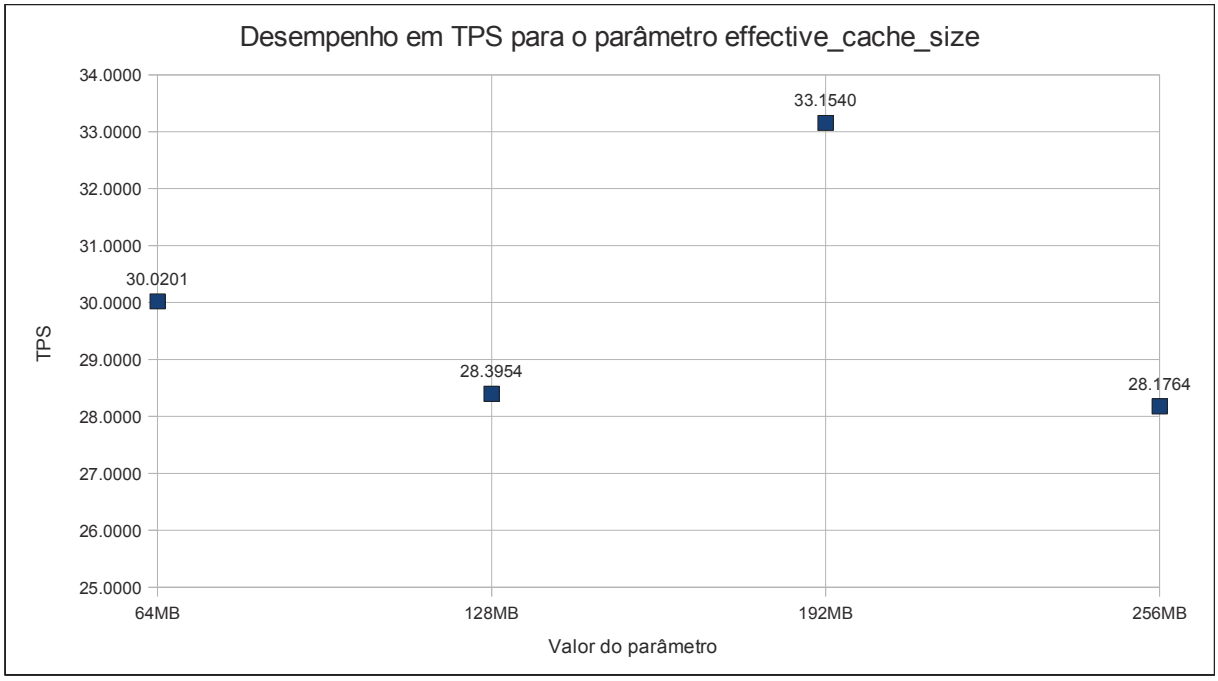

Figura 5.4: Valores fictícios de TPS encontrados na variação do parâmetro effective_cache_size

A figura 5.4 demonstra claramente a linearidade da distribuição dos valores entre os parâmetros selecionados para os trabalhos. Desta forma, percebe-se que o valor do parâmetro effective_cache_size, por ter sido distribuído de maneira linear, cobre apenas os pontos de 64MB, 128MB, 192MB e 256MB.

Conforme observado na figura 5.4, percebe-se que o maior desempenho encontrado para o SGBD durante a variação deste parâmetro ocorre quando o valor de 192MB é utilizado. Neste ponto foi encontrado o valor de 33,1540 transações por segundo, correspondendo ao máximo inicial localizado pela primeira execução do processo de pesquisa.

Devido à natureza discreta da distribuição linear dos valores utilizada, diversos pontos da área em torno do valor de 192MB para este parâmetro não são considerados. Isto faz com

que se tenha a perda de um valor próximo de 192MB para o parâmetro effective_cache_size que resultaria em um desempenho melhor ainda em transações por segundo no cenário demonstrado.

A figura 5.5 ilustra a situação descrita acima, mostrando onde ocorre a perda de um valor de pico em função da natureza discreta da distribuição linear utilizada.

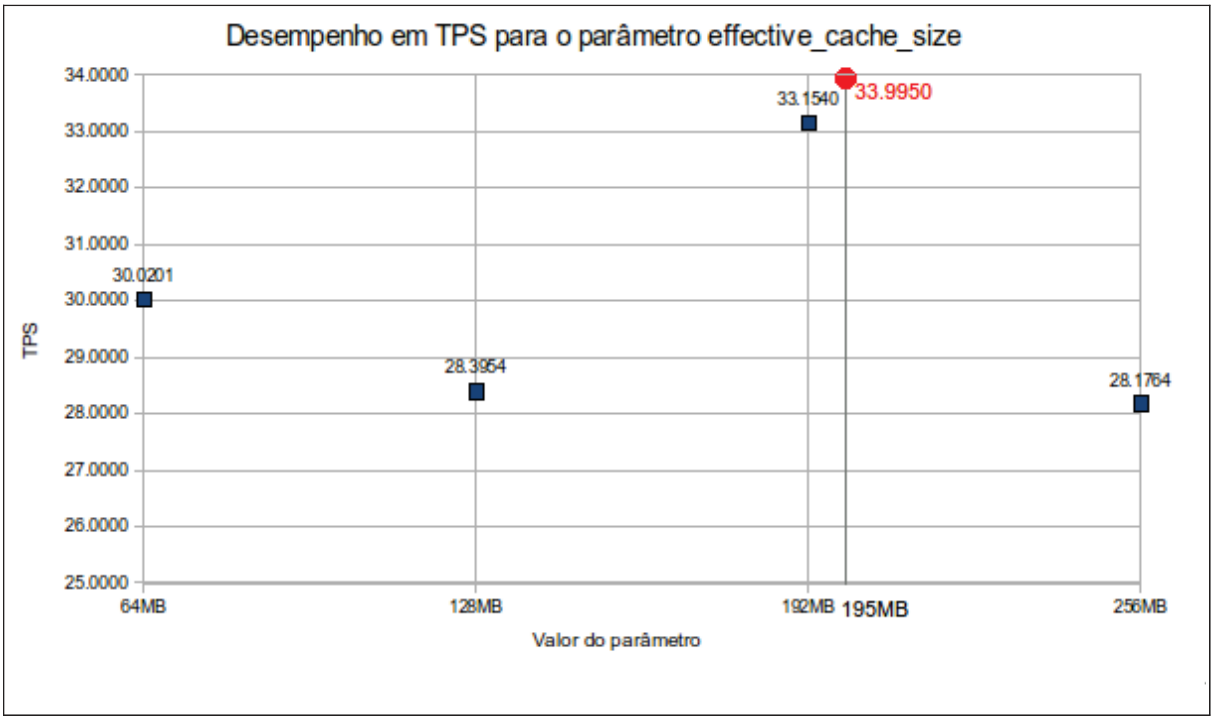

Figura 5.5: Perda de um ponto de pico em função da natureza discreta da distribuição linear

O exemplo acima ilustra o problema reduzido a apenas uma dimensão, pois foi utilizado apenas o parâmetro effective_cache_size. No entanto, o número de dimensões do problema cresce de acordo com o número de parâmetros envolvidos nos testes de desempenho. Como seis parâmetros foram utilizados, o número de dimensões do problema é o mesmo.

Para resolver o problema acima, o processo de busca de pontos ótimos foi incrementado de modo a conter também os valores para os seis parâmetros trabalhados, nas regiões próximas aos pontos de pico (máximos) previamente encontrados.

Foram selecionados os três pontos de pico para cada regime de operação utilizado, conforme exemplificado no quadro 5.2 para o regime de operação produção.

TRANSAÇÃO 1	Regime Produção					
checkpoint_segments	checkpoint_timeout	shared_buffers	effective_cache_size	wal_buffers	commit_delay	Transações por Segundo (TPS)
7	2min	14MB	192MB	96kB	4000	44.938443
7	2min	28MB	192MB	32kB	0	28.395352
5	2min	28MB	256MB	64kB	0	28.283146

Quadro 5.2: Três pontos de pico de desempenho encontrados para o regime de operação Produção

A partir dos três pontos de pico encontrados para cada um dos regimes de operação do SGBD, foi feita uma nova distribuição dos valores para os parâmetros, considerando os valores que estão 20% abaixo, 10% abaixo, 10% acima e 20% acima do valor que foi originalmente utilizado. Como pode-se observar, a nova distribuição em torno do ponto encontrado considera todas as seis dimensões, e novamente é linear para cada um dos parâmetros. Desta maneira, aumenta-se a acuracidade do número de TPS encontrado ao redor dos pontos que originalmente resultaram em pontos de pico.

O quadro 5.3 demonstra os valores utilizados para os parâmetros envolvidos nos três maiores pontos encontrados na fase preliminar de criação do modelo, bem como os novos valores que foram considerados para os testes.

Ponto 1
checkpoint_segments: 7. Intervalo Utilizado: 6, 8.
checkpoint_timeout: 2min. Intervalo Utilizado: 72s, 96s, 144s, 168s
shared_buffers: 14MB. Intervalo Utilizado: 11469kB, 12902kB, 15769kB, 17203kB,
effective_cache_size: 192MB. Intervalo Utilizado: 167MB, 180MB, 204MB, 217MB
wal_buffers: 96kB. Intervalo Utilizado: 84kB, 90kB, 102kB, 108kB
commit_delay: 4000. Intervalo Utilizado: 3200, 3600, 4400, 4800
Ponto 2
checkpoint_segments: 7. Intervalo Utilizado: 6, 8.
checkpoint_timeout: 2min. Intervalo Utilizado: 72s, 96s, 144s, 168s
shared_buffers: 28MB. Intervalo Utilizado: 25469kB, 26902kB
effective_cache_size: 192MB. Intervalo Utilizado: 167MB, 180MB, 204MB, 217MB
wal_buffers: 32kB. Intervalo Utilizado: 20kB, 26kB, 38kB, 44kB
commit_delay: 0. Intervalo Utilizado: 400, 800
Ponto 3
checkpoint_segments: 5. Intervalo Utilizado: 4, 6.
checkpoint_timeout: 2min. Intervalo Utilizado: 72s, 96s, 144s, 168s
shared_buffers: 28MB. Intervalo Utilizado: 25469kB, 26902kB
effective_cache_size: 256MB. Intervalo Utilizado: 231MB, 243MB
wal_buffers: 64kb. Intervalo Utilizado: 52kB, 58kB, 70kB, 76kB
commit_delay: 0. Intervalo Utilizado: 400, 800

Quadro 5.3: Valores utilizados na investigação de desempenho ao redor de pontos de pico

Os novos testes foram realizados no mesmo ambiente onde os testes originais haviam sido feitos. Para os três melhores pontos em cada um dos três regimes de operação, o cruzamento de todos os valores para a investigação da área ao redor dos pontos de pico resultou em um total de 43650 novos testes que foram realizados. Para cada um dos testes, novamente utilizou-se um total de 20 segundos, o que resultou em um total de aproximadamente dez dias e três horas em uma máquina dedicada a este procedimento para que estes novos testes fossem executados.

Após a execução destes testes, os resultados encontrados foram acrescentados a tabela de valores de parâmetros previamente gerada com os testes iniciais. Após a ordenação desta nova tabela pelo parâmetros TPS se obteve o modelo de otimização empregado pelo *AtTuneDB*.

6 PROTÓTIPO DE OTIMIZAÇÃO *ATTUNEDB*

Esta seção detalha o desenvolvimento do protótipo da ferramenta *AtTuneDB*, que é capaz de, utilizando os modelos definidos nas seções anteriores, inferir medidas de melhoria de desempenho para o SGBD PostgreSQL.

6.1 Requisitos e Interface do Protótipo

Para que os modelos definidos nas seções 4 e 5 pudessem ser aplicados ao ganho de desempenho em um SGBD, foi desenvolvido um protótipo para a ferramenta *AtTuneDB*.

O protótipo foi desenvolvido na linguagem Java, utilizando-se a plataforma JavaSE 1.6. O principal objetivo do protótipo *AtTuneDB* é poder, a partir de um arquivo de *log* que contenha as operações executadas por um SGBD, informar ao usuário qual o melhor conjunto de valores para os parâmetros do SGBD que afetam o seu desempenho.

Há diversos SGBDs disponíveis atualmente, sendo que muitos deles são gratuitos ou têm seu código aberto. Pela conformidade com os padrões ANSI, suporte a grandes volumes de dados, popularidade e por ser o mais avançado SGBD de código livre (PostgreSQL, 2010b), para o desenvolvimento da ferramenta detalhada nesta proposta foi escolhido o SGBD PostgreSQL. Além disto, o SGBD PostgreSQL oferece poucos recursos de auto-sintonia (MORELLI, 2006), o que faz com que uma ferramenta de apoio à sintonia deste SGBD seja bastante importante.

Diferentes passos estão envolvidos nesta tarefa, entre eles a aplicação dos modelos de identificação de regime de operação e de otimização fina do desempenho.

Em ordem de execução, tem-se como requisitos do protótipo a realização das seguintes tarefas:

- Carregamento do arquivo de *logs* informado;
- *Parsing* e normalização do arquivo carregado;
- Aplicação do modelo de identificação de regime de operação;
- Aplicação do modelo de otimização;
- Apresentação dos resultados encontrados.

As tarefas acima descritas são desempenhadas por diferentes agentes, os quais estão distribuídos em uma arquitetura autonômica apropriada para otimização constante de um SGBD PostgreSQL. Esta arquitetura será descrita em detalhes na subseção 6.3.

A Figura 6.1 ilustra a principal tela de interação com o usuário requerida para o protótipo *AtTuneDB*.

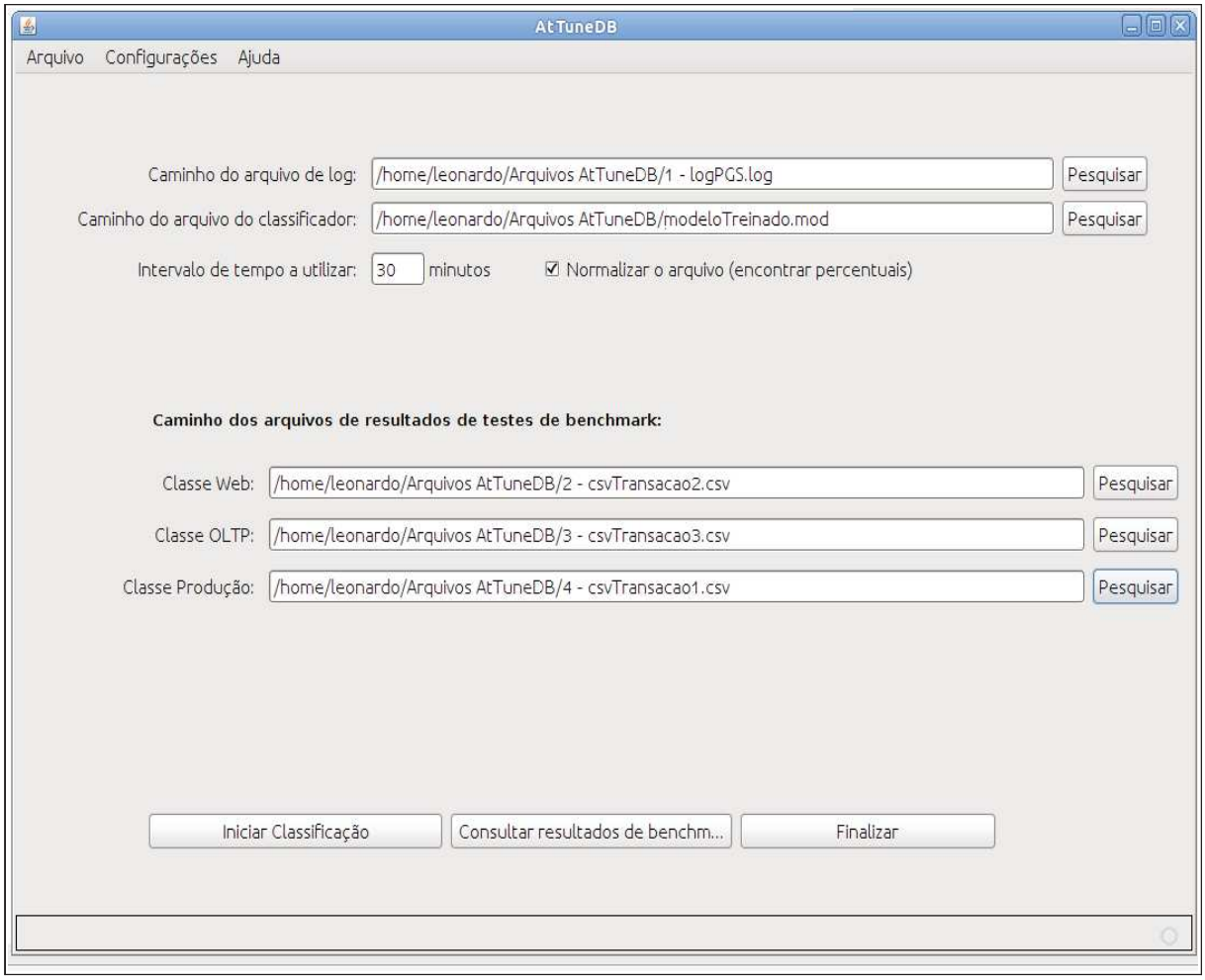

Figura 6.1: Tela principal do protótipo *AtTuneDB*

É necessário que a ferramenta tenha conhecimento de onde estão os arquivos que serão utilizados em seu funcionamento. Como pode-se perceber através da imagem, é possível informar estes dados à ferramenta.

O primeiro arquivo refere-se ao local onde se encontra o arquivo de *log* do SGBD em questão. Este será o arquivo carregado e normalizado pelo protótipo.

O processo de normalização desempenhado pelo protótipo transforma os dados lidos

do arquivo de *log* do SGBD em uma estrutura sobre a qual o modelo de identificação de regime de operação pode ser aplicado.

Através da figura 6.2 pode-se visualizar parte da estrutura de um arquivo de *log* do PostgreSQL sem nenhum tipo de *parsing* ou normalização ainda efetuados sobre o mesmo.

```
2010-09-10 01:57:12 BRT LOG:  sistema de banco de dados foi desligado em 2010-09-10 01:57:03
BRT
2010-09-10 01:57:12 BRT LOG:  pacote de inicializa??o incompleto
2010-09-10 01:57:12 BRT LOG:  sistema de banco de dados est? pronto para aceitar conex?es
2010-09-10 01:57:12 BRT LOG:  inicializador do autovacuum foi iniciado
2010-09-10 01:57:13 BRT LOG:  comando: SELECT d.datname as "Name",
               pg_catalog.pg_get_userbyid(d.datdba) as "Owner",
               pg_catalog.pg_encoding_to_char(d.encoding) as "Encoding",
               d.datcollate as "Collation",
               d.datctype as "Ctype",
               pg_catalog.array_to_string(d.datacl, E'\n') AS "Access privileges"
       FROM pg_catalog.pg_database d
       ORDER BY 1;
2010-09-10 01:57:13 BRT LOG:  comando: SELECT d.datname as "Name",
               pg_catalog.pg_get_userbyid(d.datdba) as "Owner",
               pg_catalog.pg_encoding_to_char(d.encoding) as "Encoding",
               d.datcollate as "Collation",
               d.datctype as "Ctype",
               pg_catalog.array_to_string(d.datacl, E'\n') AS "Access privileges"
       FROM pg_catalog.pg_database d
       ORDER BY 1;
2010-09-10 01:57:14 BRT LOG:  comando: SELECT d.datname as "Name",
               pg_catalog.pg_get_userbyid(d.datdba) as "Owner",
               pg_catalog.pg_encoding_to_char(d.encoding) as "Encoding",
               d.datcollate as "Collation",
               d.datctype as "Ctype",
               pg_catalog.array_to_string(d.datacl, E'\n') AS "Access privileges"
       FROM pg_catalog.pg_database d
       ORDER BY 1;
2010-09-10 01:57:28 BRT LOG:  comando: SELECT count(*) from pgbench_branches
2010-09-10 01:57:28 BRT LOG:  comando: vacuum pgbench_branches
2010-09-10 01:57:28 BRT LOG:  comando: vacuum pgbench_tellers
2010-09-10 01:57:28 BRT LOG:  comando: truncate pgbench_history
2010-09-10 01:57:28 BRT LOG:  comando: SELECT abalance FROM pgbench_accounts WHERE aid =
74319234;
2010-09-10 01:57:29 BRT LOG:  comando: SELECT abalance FROM pgbench_accounts WHERE aid =
14955764;
2010-09-10 01:57:29 BRT LOG:  comando: SELECT abalance FROM pgbench_accounts WHERE aid =
48835244;
2010-09-10 01:57:29 BRT LOG:  comando: SELECT abalance FROM pgbench_accounts WHERE aid =
66575471;
2010-09-10 01:57:29 BRT LOG:  comando: SELECT abalance FROM pgbench_accounts WHERE aid =
53310182;
```

Figura 6.2: Arquivo de *log* do PostgreSQL

O primeiro trabalho do protótipo consiste em fazer um levantamento, dentro do arquivo de *log* carregado, do número percentual de instruções *select*, *insert*, *update* e *delete* a cada instante de tempo, medido em segundos. Como instante de tempo define-se o valor informado na ferramenta no campo “Intervalo de tempo a utilizar”.

Desta forma, tem-se como saída deste processo de normalização o conteúdo exibido na figura 6.3.

```
"Tempo","Selects","Inserts","Updates","Deletes"
1,90,10,0,0
2,70,10,10,10
3,50,50,0,0
4,100,0,0,0
5,80,10,4,6
6,95,0,5,0
7,100,0,0,0
8,0,56,36,8
9,92,0,0,8
10,80,5,15,0
11,88,0,12,0
12,70,4,22,4
13,92,0,8,0
14,93,0,0,7
```

Figura 6.3: Resultado do processo de normalização

A partir do conteúdo resultante na saída acima, o protótipo *AtTuneDB* gera um arquivo CSV, o qual é utilizado como entrada para abastecer o agente que faz a aplicação do modelo de identificação do regime de operação do banco de dados. Este procedimento de aplicação do modelo está detalhado na subseção 6.2.

6.2 Aplicação dos Modelos

Após o procedimento de normalização do arquivo de *log*, o *AtTuneDB* está pronto para a aplicação dos modelos de identificação de regime de operação e de sintonia especificados nas seções 4 e 5.

A utilização destes dois modelos permite ao *AtTuneDB* encontrar um conjunto de valores otimizado para os parâmetros que possuem relação ao desempenho do SGBD PostgreSQL.

O procedimento de aplicação dos modelos está descrito nas seções 6.2.1 e 6.2.2 a seguir.

6.2.1 Aplicação do Modelo de Identificação de Regime de Operação

Para que o protótipo *AtTuneDB* tivesse a capacidade de aplicar o modelo treinado na ferramenta RapidMiner a um conjunto de dados, durante a sua implementação foi desenvolvida a integração com a ferramenta RapidMiner descrita anteriormente. Desta forma, para alcançar os resultados necessários nesta etapa, o protótipo desenvolvido segue os mesmos passos que são realizados na modelagem de um processo nesta ferramenta, ou seja, criação de um processo e a sua especificação através da criação de operadores e conexão de portas.

O processo que representa a aplicação de um modelo de classificação previamente treinado, criado na ferramenta RapidMiner, pode ser observado na Figura 6.4. Neste processo foi utilizado o modelo de identificação de regime de operação criado previamente.

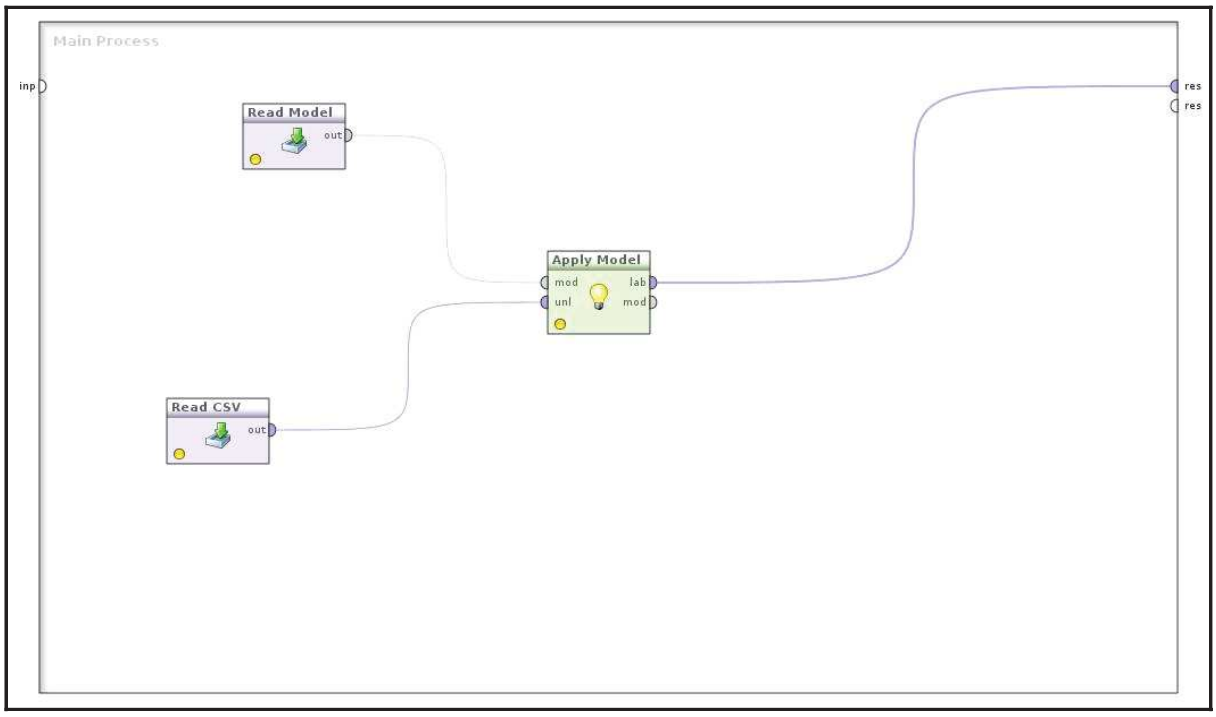

Figura 6.4: Processo para aplicação de um modelo de classificação previamente treinado.

Como pode ser observado, o processo modelado possui três operadores. O operador "Read Model" é o responsável pela leitura do arquivo contendo o modelo treinado previamente. O operador "Read CSV" responde pela leitura de um arquivo do tipo CSV, o

qual deve conter a saída normalizada descrita anteriormente e, por fim, o operador “Apply Model” é o responsável pela aplicação do modelo carregado aos dados normalizados que foram lidos.

As conexões entre os operadores deixam claro o fluxo de execução. As saídas dos operadores “Read Model” e “Read CSV” são ligadas nas entradas do operador “Apply Model”, o qual tem a sua saída ligada à saída final do processo que foi modelado.

O processo acima foi traduzido para código Java durante o desenvolvimento da ferramenta, de modo a permitir à mesma a capacidade de aplicar um modelo previamente treinado a um arquivo CSV que representa os dados normalizados. Nas propriedades dos operadores foi setado, via código, os arquivos que representam os parâmetros requeridos por estes operadores.

O procedimento descrito demonstra como *AtTuneDB* é capaz de efetuar a aplicação do modelo de identificação do regime de operação aos dados que foram normalizados nas etapas anteriores.

O resultado final da aplicação deste modelo é, justamente, a identificação de qual é o regime de operação conhecido ao modelo que mais se aproxima do real uso do SGBD. Este resultado é utilizado na próxima etapa do ciclo de operações do protótipo, a qual está descrita na subseção a seguir.

6.2.2 Aplicação do Modelo de Otimização

Após a aplicação do modelo de identificação de regime de operação, o protótipo já tem conhecimento sobre qual o regime de operação no qual foi classificado o SGBD em questão.

Neste ponto, *AtTuneDB* faz a utilização do modelo de otimização definido na seção 5.

A aplicação do modelo de otimização consiste no procedimento de varredura das tabelas deste modelo em busca de qual o conjunto de valores para os parâmetros relacionados ao desempenho que resultaram no melhor resultado para a métrica de transações por segundo nos testes de *benchmark* realizados previamente para a criação do modelo.

Após localizar os melhor valores de parâmetros de configuração para o regime de operação previametne identificado, *AtTuneDB* sugere ao usuário os valores encontrados no modelo, bem como mostra qual o regime de operação que foi identificado para o banco em

questão.

Através da figura 6.5, pode-se observar as sugestões para os valores dos parâmetros que estão sendo informadas pelo protótipo, após a aplicação do modelo de otimização.

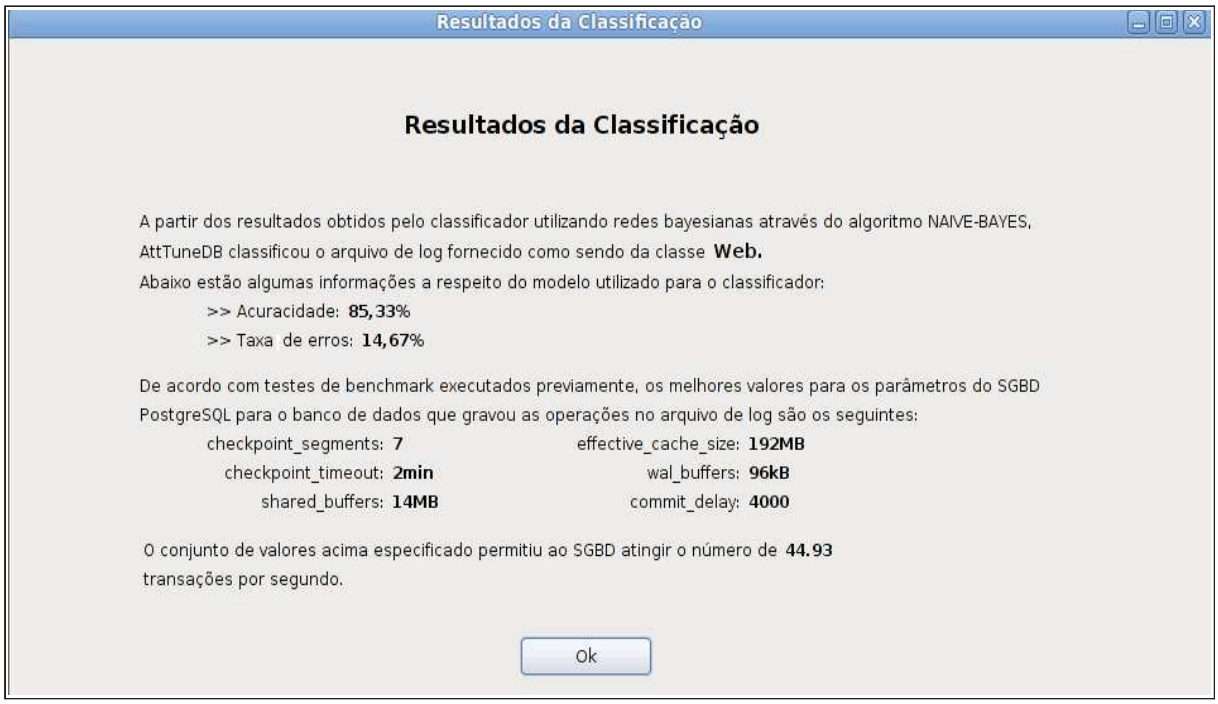

Figura 6.5: Sugestões de valores para os parâmetros

6.3 Arquitetura Autonômica

O protótipo da ferramenta *AtTuneDB* foi desenvolvido com base em uma arquitetura autonômica.

Esta subseção detalha esta arquitetura, conhecida como laço de controle autonômico, além de demonstrar como os componentes desenvolvidos no protótipo estão distribuídos através da mesma.

6.3.1 Laço de Controle Autonômico

Desenvolver um sistema com características autonômicas é uma tarefa difícil. Para amenizar este processo, foi proposta pela IBM uma arquitetura representada por um laço de controle autonômico (ver seção 2.6.5).

O laço de controle autonômico especifica que os componentes do sistema autonômico sejam divididos em componentes preestabelecidas. Estes componentes interagem entre si formando o laço.

Desta forma, o protótipo desenvolvido utiliza a arquitetura do laço de controle autonômico para resolver o problema da otimização descrito anteriormente.

6.3.2 Aplicação da Arquitetura Autonômica ao Protótipo Desenvolvido

As vantagens do uso do laço de controle autonômico pela ferramenta *AtTuneDB* permitem que o protótipo seja acoplado ao SGDB alvo (recurso gerenciado), e passe a efetuar otimizações de modo constante neste SGBD.

A arquitetura do laço de controle autonômico foi aplicada ao protótipo desenvolvido e, desta forma, criou-se os agentes necessários para a implementação da mesma.

O laço de controle autonômico descrito anteriormente consiste em uma arquitetura genérica, a qual deve ser instanciada e adaptada ao cenário que está sendo trabalhado. Desta forma, o laço de controle autonômico foi adaptado à otimização do SGBD, e a arquitetura de agentes desenvolvida para o protótipo, baseada na arquitetura autonômica, pode ser observada na figura 6.6.

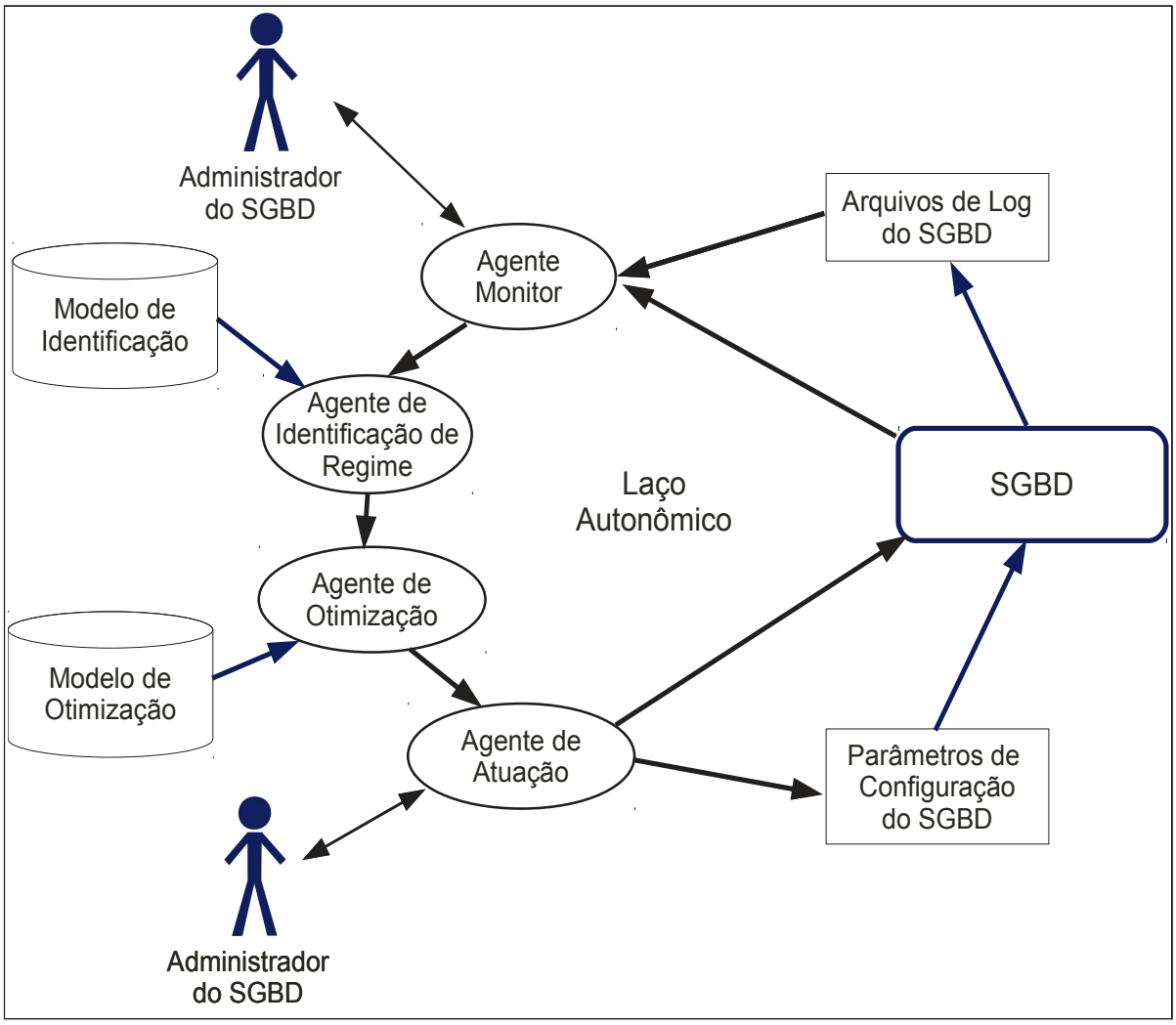

Figura 6.6: Arquitetura da ferramenta *AtTuneDB*

Os agentes criados para o protótipo incorporam os módulos e componentes principais do laço de controle autonômico utilizado.

O componente indicado na figura 6.6 pelo SGBD corresponde ao recurso gerenciado do laço de controle autonômico. Sua entrada é definida pelos parâmetros de configuração do SGBD, pois os mesmos afetam o desempenho do banco de dados e devem ser considerados como parâmetros norteadores das configurações do SGBD. Por outro lado, o SGBD fornece arquivos de *log* contendo as operações realizadas pelo mesmo, e esta é a saída deste componente para a continuidade do laço de controle autonômico.

Como pode ser constatado a partir da figura 6.6, o agente de identificação de regime de operação do SGBD corresponde ao componente analisador do laço de controle autonômico, enquanto que o agente de otimização corresponde ao componente planejador do laço de controle autonômico. Estes dois agentes foram implementados em sua totalidade no

protótipo desenvolvido, sendo que os mesmos incorporam os modelos de identificação de regime de operação e de otimização do SGBD descrito nas seções 4 e 5 para desempenharem suas funções.

A implementação dos agentes de monitoração e de atuação, identificada na arquitetura autonômica como monitores e atuadores foi simplificada no protótipo *AtTuneDB*. O agente de monitoração oferece um mecanismo básico de acesso aos arquivos de *log* fornecidos pelo SGBD trabalhado. Este é o recurso que o agente monitora para poder fornecer insumos ao próximo componente do laço, a saber, o agente analisador (identificação do regime de operação). Já o agente de atuação concentra a interface com o usuário, a qual fornece um *feedback* a respeito das decisões encontradas pelos agentes pertencentes ao laço autonômico.

O administrador do SGBD é um componente que aparece na arquitetura proposta pois, de acordo com a implementação do protótipo definida anteriormente, é necessária a sua intervenção em dois momentos para que o laço de controle autonômico possa fluir. Em um primeiro momento, o DBA precisa informar onde estão os arquivos contendo os *logs* do banco, os quais serão utilizados pelo agente analisador. Num próximo passo, o DBA também estará conectado ao agente atuador, pois o que a ferramenta criada disponibiliza são os valores ideais para um melhor desempenho do SGBD, não efetuando a alteração destes valores de forma automática pelos motivos que serão discutidos a seguir.

Todos os agentes descritos foram criados e testados em um ambiente controlado. Ressalta-se que em um sistema real estes agentes seriam mais complexos, uma vez que geração de *logs* pelo próprio SGBD é uma tarefa onerosa e que deve conter um conjunto mínimo de informações para que não consuma em excesso os recursos computacionais disponíveis no ambiente onde o SGBD se encontra. Por outro lado, a modificação dos valores dos parâmetros do SGBD deve ser realizada com bastante cuidado, pois o funcionamento do SGBD depende desta configuração. Também é importante ressaltar que o SGBD trabalhado (PostgreSQL) precisa ser reiniciado após a alteração de valores para os seus parâmetros, o que restringe a faixa de horários em que a aplicação possa trabalhar, visto que, na grande maioria dos casos, o SGBD permanece operando durante longos períodos de tempo com vários usuários conectados ao mesmo, não tornando possível que se pare o mesmo a qualquer tempo.

As funcionalidades do protótipo desenvolvido estão mapeadas, para os componentes do laço de controle autonômico, de acordo com os detalhamentos encontrados nas subseções a seguir.

6.3.2.1 Agente de Identificação de Regime

Na arquitetura da ferramenta *AtTuneDB*, o agente de Identificação de Regime implementa os componentes analisadores previstos na arquitetura autonômica genérica. Este agente é responsável por efetuar a análise do arquivo de *log* que representa as operações executadas no SGBD.

Em um primeiro momento, este agente é responsável por efetuar o *parsing* no arquivo de *log* fornecido de modo a utilizar apenas as linhas que se referem às transações de *select*, *insert*, *update* e *delete* que foram executadas pelo SGBD. Isto é necessário para que a ferramenta consiga classificar corretamente o regime de operação sendo utilizado pelo SGDB que gerou o *log*.

Como resultado do procedimento de *parsing* tem-se uma saída normalizada, a qual representa o percentual correspondente à quantidade de cada uma das operações de *select*, *insert*, *update* e *delete* fornecidas no arquivo de *log*.

Neste ponto, o agente está pronto para efetuar a aplicação do modelo de identificação do regime de operação, o qual foi explanado na seção 4. O procedimento de aplicação do modelo de identificação de regime de operação consiste na análise da saída normalizada e, com base no raciocínio probabilístico fornecido pelo modelo treinado, efetuar a classificação do uso do SGBD em um dos regimes de operação descritos na seção 4.1.1.

A aplicação do modelo de identificação de regime de operação realizada neste momento foi detalhada na seção 6.2.1.

6.3.2.2 Agente de Otimização

O agente de Otimização tem a tarefa de definir a melhor solução para o problema da otimização. Este agente da *AtTuneDB* implementa os componentes planejadores previstos na arquitetura autonômica genérica, que são responsáveis por avaliar qual o melhor conjunto de valores para os parâmetros do SGDB em questão.

O resultado das tarefas realizadas pelo agente de Identificação de Regime consiste na classificação do regime de operação do SGBD. O agente de Otimização se baseia nesta classificação recebida do agente de Identificação para pesquisar, nos resultados de *benchmark*

de desempenho efetuados anteriormente, qual foi o conjunto de valores de parâmetros do SGBD cujo teste resultou no melhor desempenho para o regime de operação definido.

Esta tarefa configura a utilização do modelo de otimização criado em ambiente controlado no desenvolvimento da ferramenta, o qual encontra-se definido na seção 5.

Uma vez encontrados, durante a utilização do modelo, os valores que geraram o melhor desempenho no teste de *benchmark*, cabe ao agente de otimização a tarefa de recuperação destes valores, para que os componentes executores definidos na seção seguinte possam seguir o fluxo definido pelo laço de controle autonômico.

A aplicação do modelo de otimização realizada neste momento foi detalhada na seção 6.2.2.

6.3.2.3 Agente de Atuação

No estado atual do protótipo desenvolvido, os executores têm a tarefa de informar qual o regime de operação identificado, bem como o conjunto de valores dos parâmetros que resultou no melhor desempenho do SGBD.

Somente informar os valores é uma limitação do protótipo desenvolvido, pois o comportamento ideal incluiria intervenções diretamente nas configurações do SGBD para que o mesmo fosse reconfigurado para um melhor desempenho sem que o administrador do SGBD precisasse interferir neste processo.

Pelo fato de se tratar de um protótipo, a interface com o usuário foi a maneira de trabalho definida para os componentes atuadores deste protótipo.

6.3.2.4 Agente de Monitoração

O agente de monitoração é responsável por monitorar a operação do SGBD, coletando registros estatísticos a respeito das operações sendo executadas pelo SGBD. No protótipo do *AtTuneDB* este agente implementa uma interface de acesso aos arquivos de *log* que devem ser gerados pelo próprio SGBD. Os arquivos de *log* fornecidos pelo SGBD denotam o insumo necessário para os trabalhos deste agente.

O agente de monitoração implementado provê uma interface ao usuário, a qual permite definir onde estão armazenados os arquivos de *log* do SGBD. Esta interface pode ser observada na figura 6.1.

7 EXPERIMENTOS E RESULTADOS OBTIDOS

Esta seção descreve os experimentos que foram realizados para a validação da ferramenta desenvolvida, e apresenta os resultados obtidos com os experimentos.

7.1 Cenários Utilizados nos Experimentos

Para os experimentos que foram realizados com o objetivo de validação do protótipo foi utilizado o seguinte ambiente computacional:

- Sistema Operacional Linux Ubuntu 10.04;
- Processador Intel Core2Duo, clock de 1.66Ghz;
- Memória de 2 gigabytes;
- Disco Rígido com 160 gigabytes, SATA;
- SGBD PostgreSQL versão 8.4.

Em um primeiro momento, foi instalado o SGDB PostgreSQL onde serão executados os experimentos. Foi realizada uma instalação padrão, sem alteração de qualquer parâmetro, ou seja, a instalação foi feita conforme as determinações do fabricante, sem qualquer ajuste. Este cenário configura uma situação onde não há sintonia do SGBD para o aproveitamento de qualquer recurso disponível, o que faz com que o desempenho não esteja otimizado para o ambiente onde o SGDB está instalado. Também foi instalada a ferramenta pgbench, responsável pelos testes de benchmark.

No SGBD recém-instalado foram populadas tabelas fictícias, conforme os procedimentos descrito no item 5.2.1. O processo de população do SGBD com uma massa de dados significativa faz com que os testes apresentem resultados mais próximos da realidade encontrada nos SGBDs atuais. Após o SGBD estar populado com os dados desejados, iniciaram-se os testes.

Para cada um dos três regimes de operação trabalhados pela ferramenta, foram executados por cinco vezes os testes de *benchmark* contendo os conjuntos de instruções que caracterizam estes regimes. Após a execução das cinco vezes, foi calculada a média de transações por segundo atingidas com o SGBD em sua configuração padrão, sem qualquer

alteração.

O quadro 7.1 demonstra as chamadas feitas para que o aplicativo pgbench executasse os testes de *benchmark*, mostrando as opções que foram utilizadas e caracterizando os testes que foram realizados.

Regime de Operação	Chamada Executada
Web	./pgbench -S -T 20
OLTP	./pgbench -N -T 20
Produção	./pgbench -T 20

Quadro 7.1: Chamadas Realizadas para os Testes

Os testes executados para medição do número de transações por segundo seguiu o padrão dos testes realizados para abastecimento do modelo de otimização do SGBD descrito na seção 6, por se tratar, conforme descrito na referida seção, de um padrão sugerido pelo fabricante da ferramenta.

A opção -T 20 informa à ferramenta de *benchmark* a solicitação para que a mesma permaneça executando o mesmo teste por 20 segundos, para assim obter um resultado mais preciso do que se fosse executado apenas um único teste para o regime de operação em questão.

Foram coletados os resultados apresentados para as execuções das chamadas apresentadas no quadro 7.1.

O quadro 7.2 demonstra os resultados obtidos, em transações por segundo, após a realização dos testes descritos.

Regime de Operação	Rodagem	TPS alcançado
Web	1	28,4412
	2	30,1541
	3	30,4033
	4	31,6067
	5	29,4803
	Média Web	30,0171
OLTP	1	23,0421
	2	21,0382
	3	23,5768
	4	21,0426
	5	23,2198
	Média OLTP	22,3839
Produção	1	21,7492
	2	21,7141
	3	22,9296
	4	18,0292
	5	23,4741
	Média Produção	21,5792

Quadro 7.2: Resultados obtidos antes da sintonia

7.2 Resultados Obtidos

Após a execução dos testes nos cenários descritos, foi utilizado o protótipo desenvolvido com o intuito de melhorar o desempenho do SGBD em questão.

A primeira tarefa que coube à ferramenta foi a classificação de cada um dos regimes de operação trabalhados. A ferramenta foi capaz de classificar corretamente cada um dos regimes, o que validou o modelo de identificação de regimes de operação descrito na Seção 4.

Após a detecção de cada um dos regimes de operação nas transações executadas, o protótipo *AtTuneDB* utilizou-se do modelo de otimização apresentado na Seção 5 para sugerir o melhor conjunto de valores para os parâmetros do SGBD que otimizasse seu desempenho.

Para cada um dos regimes de operação trabalhados, o serviço SGBD foi parado. Neste

momento, abriu-se o arquivo de configurações do SGBD (*postgresql.conf*) e os valores neste arquivo foram alterados conforme os valores sugeridos pela ferramenta.

Após isto, iniciou-se novamente o serviço do SGBD, sendo que neste momento o mesmo passou a utilizar os novos valores para os parâmetros que foram informados no arquivo de configurações.

Novamente foram executados os mesmos testes realizados antes da sintonia dos parâmetros (descritos no quadro 8), para que o novo número de transações por segundo fosse auferido, já com os valores alterados, podendo assim chegar-se a alguma conclusão em relação às vantagens do uso do protótipo desenvolvido para a melhoria de desempenho do SGBD.

O quadro 7.3 apresenta os resultados dos novos testes, com o SGBD já sintonizado em relação aos seus parâmetros.

Regime de Operação	**Rodagem**	**TPS Anterior**	**Novo TPS**
Web	1	28,4412	32,7854
	2	30,1541	34,3752
	3	30,4033	34,6139
	4	31,6067	33,3070
	5	29,4803	33,0456
	Média Web	30,0171	33,6254
OLTP	1	23,0421	24,3986
	2	21,0382	24,8938
	3	23,5768	24,6949
	4	21,0426	22,3453
	5	23,2198	24,7366
	Média OLTP	22,3839	24,2138
Produção	1	21,7492	23,6588
	2	21,7141	23,8211
	3	22,9296	24,0207
	4	18,0292	23,7824
	5	23,4741	24,0810
	Média Produção	21,5792	23,8728

Quadro 7.3: Resultados obtidos antes e após da sintonia

Para uma melhor comparação, o quadro 7.4 demonstra, em percentual, o ganho real que foi obtido com o uso da ferramenta *AtTuneDB* em cada um dos regimes de operação trabalhados.

Regime de Operação	Ganho em Percentual
Web	12,02%
OLTP	8,18%
Produção	10,63%

Quadro 7.4: Percentuais de Ganho com o uso de *AtTuneDB*

Os resultados demonstrados no quadro 7.4 validam o modelo de otimização desenvolvido para a ferramenta, pois mostram percentualmente que, para todos os regimes de operação trabalhados, o desempenho do SGBD foi melhorado com o uso do protótipo. Também ilustra o sucesso da ferramenta no fato de se conseguir uma sintonia fina dos parâmetros do SGBD que forneça um ganho de desempenho real.

Além dos testes realizados com os regimes de operação acima definidos, o modelo de identificação de regime de operações também foi validado de outra forma. Dada a dificuldade de se encontrar logs reais de SGBDs que estão em produção, visto os mesmos dificilmente gravarem em log as suas operações de inserção, seleção, deleção e atualização, partiu-se para a obtenção de um arquivo de log correspondente a uma aplicação cujo regime de operação seja conhecido previamente.

Para tanto, foi desenvolvido um sítio composto por três páginas com ligações entre si para que haja a navegação entre as mesmas, através da ferramenta Joomla. Joomla é um sistema gerenciador de conteúdo (CMS) para a criação de sítios na internet, e foi utilizado por ser um dos mais populares CMS da atualidade, com uma comunidade atuante e crescente (Joomla, 2010).

Neste sítio criado foram feitos acessos de forma esporádica durante uma semana, e após este tempo foi coletado o log do SGBD PostgreSQL selecionado para mantê-lo. O arquivo de log coletado foi submetido à ferramenta desenvolvida e a mesma obteve êxito em classificar o mesmo como sendo do regime de operação Web. Desta forma, o modelo foi validado mais uma vez, além das validações realizadas com os arquivos de log gerados a partir dos testes com a ferramenta pgbench.

8 CONCLUSÕES

O desempenho de um SGBD é um requisito essencial deste tipo de sistema, sendo considerado desde a definição inicial do SGBD a ser utilizado por determinada aplicação, até o momento em que o mesmo estará em produção servindo à uma determinada solução. Atividades para se obter e manter um desempenho otimizado por parte do SGBD são esforços que devem ser dispendidos continuamente, a fim de garantir o desempenho ótimo do sistema.

A atividade conhecida como sintonia fina de um SGBD pode prover ganhos de desempenho sem que para isto sejam necessários investimentos em hardware, ou seja, o SGBD pode ter o seu desempenho otimizado e com isto responder melhor às solicitações sem que haja uma mudança significativa no ambiente computacional físico envolvido.

Este trabalho demonstrou em laboratório a viabilidade da integração de técnicas de mineração de dados, modelos probabilísticos de classificação e métodos de pesquisa em espaços de configuração, no contexto de uma arquitetura autonômica de agentes, para que se possa efetuar uma sintonia fina em um SGBD PostgreSQL e desta maneira obter-se um desempenho otimizado do mesmo.

Na seção 3 pode se observar que, apesar de diversos esforços da comunidade científica para a correta sintonia de um SGBD, o uso integrado das técnicas envolvidas neste trabalho não havia sido empregado ainda.

Conclui-se, desta maneira, que não só essa integração de técnicas é possível, mas a ferramenta resultante, implementada na forma de um protótipo, efetivamente é capaz de melhorar o desempenho de um SGBD, pelo menos nas situações controladas de laboratório. O uso de agentes de *software* para a implementação da mesma flexibiliza a solução e permite que o laço de controle autonômico seja utilizado, aproveitando-se assim a ferramenta de uma arquitetura estabelecida.

8.1 Principais Contribuições

Como principais contribuições desta pesquisa, destacam-se os seguintes itens:

- Divisão do processo de otimização entre a identificação do regime de operação do SGBD e a posterior busca dos valores ótimos para seus parâmetros de

configuração.

- Uso de técnicas de mineração de dados com um classificador bayesiano simples para identificação do regime de operação do SGBD. Essas combinação particular de técnicas e modelos não havia sido empregado anteriormente para a obtenção de uma sintonia fina de parâmetros de um SGBD, e mostrou-se eficaz quando foi utilizado como modelo de identificação de regime de operações do SGBD.
- Criação de um modelo de otimização de parâmetros de configuração baseado na pesquisa de valores ótimos orientada por regime de operação do SGBD, através de um método de pesquisa desses valores que usa mecanismos não-lineares de busca a fim de permitir refinamentos importantes nos valores encontrados.
- Proposta de uma arquitetura de agentes baseada na arquitetura de laço de controle autonômico, originalmente especificada pela IBM.

8.2 Trabalhos Futuros

O principal foco do presente trabalho foi na criação e validação de modelos apropriados para o núcleo do laço autonômico, constituído pelas as etapas de análise e planejamento das operações de otimização do SGBD.

Entretanto, para que o laço de controle autonômico possa ser concluído e utilizado em sua totalidade, alguns esforços são necessários para preencher requisitos que não foram implementados de forma completa no protótipo da ferramenta aqui proposta.

Na ferramenta, o passo mais importante para o uso do laço de controle autonômico foi dado. Porém, é necessário que seja feita uma reengenharia das funções que são cabidas aos agentes responsáveis pelos componentes monitores e atuadores do laço de controle autonômico. Fazer com que a ferramenta esteja acoplada ao SGBD que está sendo utilizado com a mesma traz benefícios no ganho de desempenho e, principalmente, nos níveis de autonomia da ferramenta.

Com a ferramenta acoplada ao SGBD, os agentes responsáveis pelos componentes atuadores podem efetuar ou ativar mudanças no SGBD de acordo com decisões tomadas unicamente pela ferramenta, fazendo com que, desta forma, não seja requerida uma ação do

DBA para que se otimize o desempenho do SGBD, sendo a própria ferramenta a responsável pela melhoria do desempenho. Isto faria com que o DBA tivesse seu tempo disponível para outras atividades que não a sintonia do SGBD, podendo assim atingir de forma mais efetiva suas metas de trabalho.

Para que isto seja alcançado, há a necessidade da extensão deste trabalho, através da extensão e generalização dos mecanismos de monitoração e atuação implementados no protótipo, para que efetivamente trabalhem ligados ao SGBD que está sendo utilizado. Os novos agentes que implementarão estes mecanismos, irão monitorar o SGBD de forma automática, permitindo que o mesmo, sendo o recurso gerenciado do laço de controle autonômico, disponha de recursos de auto-otimização (um dos conceitos-chave da computação autonômica) para assim melhorar o seu desempenho sem necessitar de intervenção humana.

REFERÊNCIAS

BORDINI, Rafael et al. **Multi-Agent Programming Languages, Platforms and Applications**. Springer, Nova York, 2005.

BORDINI, Rafael; HÜBNER, Jomi; WOOLDRIDGE, Michael. **Programming Multi-Agent Systems in AgentSpeak using Jason**. 2007. ISBN: 970-0-470-02900-8.

BORDINI, Rafael; VIEIRA, Renata. Linguagens de programação orientadas a agentes: uma introdução baseada em AgentSpeak(L). **Revista de Informática Teórica e Aplicada**, v. 10, p. 7–38, Agosto, 2003. Instituto de Informática da UFRGS, Brasil.

BRAGA, Antônio de Pádua; LACERDA, Wilian Soares. Experimento de um Classificador de Padrões Baseado na Regra Naive de Bayes. **Infocomp: Journal of Computer Science**, v. 3, n. 1, p. 30-35, 2004.

BRATMAN, Michael. **Intentions, plans and practical reason**. Cambridge, MA: Harvard University Press, 1987.

CHARNIAK, Eugene. Bayesians Networks without Tears. **IA Magazine**, v. 12, n. 4, p. 50-63, 1991.

CHATZIDIMITRIOU, Kyriakos; SYMEONIDIS, Andreas. Data-Mining-Enhanced Agents in Dynamic Supply-Chain-Management Environments. **IEEE Intelligent Systems,** vol. 24, no. 3, p. 54-63, 2009.

CASTILLO, Enrique; GUTIÉRRES, Jose Manuel e HADI, Ali. **Sistemas Expertos y Modelos de Redes Probabilísticas.** Madri: Academia Española de Ingeniería, 1998.

COOPER, Gregory. The computational complexity of probabilistic inference using Bayesian belief networks. **Artificial Intelligence**, vol. 42, no. 2, p. 393-405, 1990.

COSTA, Rogério Luis Carvalho et al. Implementation of an Agent Architecture for Automated Index Tuning. **Proceedings of the ICDE Workshops**. p. 1215, 2005.

DATE, Christopher. **Introdução a sistemas de bancos de dados**. Rio de Janeiro: Campus, 2000.

DEBNATH, Biplob; LILJA, David; MOKBEL, Mohamed. SARD: A statistical approach for ranking database tuning parameters. **Proceedings of the 2008 IEEE 24th International Conference on Data Engineering Workshop**, p.11-18, 2008

DENNETT, Daniel. The intentional stance. Cambridge, MA: The MIT Press, 1987.

DUAN, Songyun; THUMMALA, Vamsidhar, BABU, Shivnath: Tuning Database Configuration Parameters with iTuned. **PVLDB 2**, vol. 1, p. 1246-1257, 2009.

DUTRA, Inês. Disponível em <http://www.cos.ufrj.br/~ines/courses/cos740/leila/cos740/Bayesianas.pdf>. Acesso em 02 abril 2010.

ELNAFFAR, Said et al. Today's DBMS: How autonomic are they? **14th International Workshop on Database and Expert Systems Applications**, Kingston, p. 651-544, 2003.

FAYYAD, Usama; PIATETSKY-SHAPIRO, Gregory; SMYTH, Padhraic. **From data mining to knowledge discovery: Advances in knowledge discovery & Data Mining**, Menlo Park, 1996.

GLUZ, João Carlos; FLORES, Cecilia; VICARI, Rosa Maria. Formal Aspects of Pedagogical Negotiation in AMPLIA System. In: Nadia Nedja; Luiza M. Mourelle; Nival N. de Almeida; Mario N. Borges. (Org.). **Intelligent Educational Machines**. New York: Springer, 2006, v. 44, p. 117-146.

HAN, Jiawei; CHANG, Kevin. Data Mining for Web Intelligence. **Computer**, v. 35, n. 11, p. 64-70, 2002.

HENDERSON-SELLERS, Bryan; GIORGINI, Paolo. **Agent-Oriented Methodologies**. Idea Group, Hershey, PA, 2005.

HORN, Paul. **Autonomic Computing:** IBM's perspective on the state of information technology. Nova Iorque: IBM PRESS, 2001.

JACOB, Bart et al. **A practical guide to the IBM autonomic computing toolkit.** IBM Red Books, 2004.

Joomla. Disponível em: <http://www.joomla.com.br> Acesso em: 12 dez. 2010.

Linux Depot. Disponível em: <http://www.linuxdepot.com.br/otimizando_postg resql.txt> Acesso em: 11 jul. 2008.

LOPES, Lucelene. **Aprendizagem de Máquina Baseada na Combinação de Classificadores em Bases de Dados da Área da Saúde.** Curitiba: PUC-PR, 2007.

LOZANO, Fernando. Disponível em: <www.lozano.eti.br/palestras/tunning-pgsql.pdf> Acesso em: 05 Junho 2010.

MACHADO, Leonardo Ribeiro; NASCIMENTO, Francisco Assis. ADA - an autonomic agent for self-tuning in PostgreSQL databases. **LAACS'08 - Latin American Autonomic Computing Symposium**, v. 3, n. 1, 2008.

MANOEL, Edson et al. **Problem determination using self-managing autonomic technology.** IBM Red Books, 2005.

MARKL, Volker. LEO: An autonomic query optimizer for DB2. **IBM Systems Journal**, n. 42, p. 98-103, 2003.

MARKOV, Daniel; LAROSE, Zdravko. **DATA MINING THE WEB: Uncovering Patterns in Web Content, Structure, and Usage**. 1. ed. Nova Jersey: John Wiley & Sons, 2007

MITCHELL, Tom. **Machine Learning**. Nova Iorque: McGraw-Hill, 1997.

MOMJIAN, Bruce. PostgreSQL Performance Tunning. **Linux Journal**, v. 88, 2001.

MORELLI, Eduardo et al. **Reindexação Automática em SGBDs Relacionais.** 2009. XXIV Simpósio Brasileiro de Banco de Dados (SBBD).

MORELLI, Eduardo. **Recriação Automática de Índices em um SGBD Relacional**. Rio de Janeiro: PUC-Rio, 2006.

MURCH, Richard. **Autonomic computing**. Nova Iorque: IBM Press, 2004.
NAVEGA. Disponível em: <http://www.intelliwise.com/reports/info2001.htm> Acesso em: 07 nov. 2007

PARASHAR, Manish; HARIRI, Salim. **Autonomic computing:** concepts, infrastructure and applications. Boca Raton: CRC Press, 2007.

PGBench. Disponível em <http://developer.postgresql.org/pgdocs/postgres/pgbench.html> Acesso em: 07 Agosto 2010.

PostgreSQL. Disponível em: <http://www.postgresql.org/> Acesso em: 17 jun. 2010.

PostgreSQL. Disponível em:<http://www.postgresql.org/docs/current/static/> Acesso em: 17 jun. 2010.

POWLEY, Wendy et al. **Autonomic buffer pool configuration in PostgreSQL**. IEEE Systems, Man and Cybernetics, p. 53-58, out. 2005.

Rapid-I. Disponível em <http://rapid-i.com/>. Acesso em 02 Jul. 2010.

RUSSEL, Stuard; NORVIG, Peter. **Artificial Intelligence A Modern Approach. Second Edition.** Prentice Hall. 2003.

SALLES, Marcos A. V.; LIFSHITZ, Sérgio. Um agente de software para a criação de índices no PostgreSQL. **Brazilian Symposium on Databases (SBBD)**, n. 1, p. 37-42, 2004.

SHASHA, Dennis; BONNET, Phillipe. **Database Tuning: Principles, Experiments and Troubleshooting Techniques.** San Francisco: Morgan Kaufmann, 2003.

SILBERSCHATZ, Abraham; FORTH, Henry; SUDARSHAN, S. **Sistema de banco de dados**. 3. ed. São Paulo: Makron Books, 1999.

TAN, Pang-ning; STEINBACH, Michael; KUMAR, Vipin. **Introdução ao data mining - Mineração de Dados**. 1. ed. Rio de Janeiro: Ciência Moderna, 2009.

TPC. Disponível em: <http://www.tpc.org/> Acesso em: 02 Agosto 2010

TPC-H. Disponível em: <http://www.tpc.org/tpch/> Acesso em: 06 Junho 2010

VICARI, Rosa Maria.; GLUZ, João Carlos. An Intelligent Tutoring System (ITS) View on AOSE. **International Journal of Agent-Oriented Software Engineering (IJAOSE**) (Print). , v.1, p.295 - 333, 2007.

VICARI, Rosa Maria et al. AMPLIA: A Probabilistic Learning Environment. **International Journal of Artificial Intelligence in Education**, v. 18, p. 347-373, 2008.

WEI, Li et al. A Practical Tool for Visualizing and Data Mining Medical Time Series. **CBMS 2005**, p. 341-346, 2005.

WEISS, Gehrard. **Multiagent systems: A modern approach to distributed artificial intelligence**. Londres: MIT Press, 1999.

WIESE et al. Autonomic Tuning Expert – A framework for best-practice oriented autonomic database tuning. **Proceedings of the 2008 CASCON Conference**, Ontario, 2008.

WOOLDRIDGE, Michael. **An Introduction to Multiagent Systems**. John Wiley and Sons, Chichester, England, 2002.

WOOLDRIDGE, Michael. **Reasoning about rational agents**. Cambridge, Massachusetts: MIT Press, 2000.

ZHANG, Harry. The optimality of Naive Bayes. **Proceedings of the 17th International FLAIRS Conference.** Florida, 2004.