FACULDADES INTEGRADAS DE TAQUARA
CURSO DE SISTEMAS DE INFORMAÇÃO

ADA: UM AGENTE AUTONÔMICO PARA AUTO-SINTONIA EM BANCOS DE DADOS POSTGRESQL

LEONARDO RIBEIRO MACHADO

Taquara

2008

LEONARDO RIBEIRO MACHADO

ADA: UM AGENTE AUTONÔMICO PARA AUTO-SINTONIA EM BANCOS DE DADOS POSTGRESQL

Trabalho de Conclusão apresentado ao Curso de Sistemas de Informação das Faculdades Integradas de Taquara, como requisito parcial para a obtenção do grau de Bacharel em Sistemas de Informação, sob orientação do Professor Me. Francisco Assis Moreira do Nascimento.

Taquara

2008

RESUMO

Com a crescente complexidade das aplicações que utilizam Sistemas Gerenciadores de Bancos de Dados (SGBDs) e levando-se em conta a diversidade da carga utilizada por estas aplicações, soluções autonômicas são cada vez mais necessárias para satisfazerem problemas complexos dos SGBDs atuais. A otimização de um SGBD é uma tarefa complexa de ser realizada por exigir que sejam considerados diversos fatores que podem inferir na correta configuração dos parâmetros que podem afetar o desempenho do SGBD, como análise dos recursos de *hardware* disponíveis e variáveis de ambiente. Para tanto, é proposta a implementação do agente ADA, uma ferramenta capaz de autonômica e gradualmente sintonizar, de maneira otimizada, os parâmetros do SGBD PostgreSQL que possuem relação com o seu desempenho, levando em conta a carga real que está sendo submetida ao banco e o ambiente computacional onde o mesmo está inserido, com a finalidade de melhorar a sua performance.

Palavras-chave: Computação Autonômica. Sistemas Gerenciadores de Bancos de Dados. Auto-otimização. PostgreSQL.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE SIGLAS

ADA - *Autonomic Database Agent*

API - *Application Programming Interface*

DBA - *Database Administrator*

IDE - *Integrated Development Environment*

JDBC - *Java Database Connectivity*

LEO - *Learning Optmizer*

OLTP - *Online Transaction Processing*

RAM - *Random-access memory*

SQL - *Structured Query Language*

TPC - *Transaction Processing Performance Council*

TPS - Transações por Segundo

WAL - *Write-Ahead Log*

XML - *Extensible Markup Language*

SUMÁRIO

1 INTRODUÇÃO

O presente projeto demonstra a aplicação de técnicas de computação autonômica como norteadores para a configuração dos parâmetros disponíveis em um Sistema Gerenciador de Banco de Dados (SGBD). Detalha todo o processo envolvido na construção de uma ferramenta capaz de autonomicamente inferir mudanças responsáveis por melhorias nas operações do SGBD, com especificações acerca das ferramentas e técnicas utilizadas, demonstrando como os resultados foram obtidos e a importância dos itens utilizados para que o objetivo final fosse alcançado.

A computação autonômica é uma área relativamente nova. O termo surgiu no ano de 2001, introduzido pelo vice-presidente de pesquisas da IBM, Paul Horn, e se refere a sistemas que podem gerenciar a si próprios a partir de objetivos de alto nível determinados por administradores. Desta maneira, faz uma alusão ao sistema nervoso autonômico, presente no organismo de todos os seres humanos (HORN, 2001).

A Figura 1 exemplifica o funcionamento do sistema nervoso autonômico: ao passo em que o homem está lendo seu jornal, seu sistema nervoso autonômico é responsável por executar uma série de tarefas. Estas tarefas são realizadas inconscientemente, sem a necessidade de o homem precisar pensar nelas para que as mesmas sejam realizadas, como digerir o café da manhã e ajustar a taxa de respiração, por exemplo. Da mesma maneira, a computação autonômica é responsável por uma série de melhorias que podem ser aplicadas a um sistema, sem que estas melhorias sejam fonte de preocupação constante por parte dos usuários desse sistema.

Figura 1: O Sistema Nervoso Autonômico em Ação
Fonte: HORN (2001).

A computação autonômica é um grande desafio a ser alcançado pelas empresas, e um esforço muito grande deve ser empenhado para que se consiga embutir esta característica em novos sistemas e também aos sistemas que hoje já se conhece (KEPHART e CHESS, 2001).

Este trabalho aplica técnicas de computação autonômica ao SGBD PostgreSQL. Com a massificação dos SGBDs através de seu freqüente uso por entidades que possuem sistemas computadorizados, cresce também a necessidade de se manter o correto e adequado funcionamento dos mesmos. 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 para manter esta estrutura torna-se importante, pois cada vez mais se necessita de um correto monitoramento do SGBD com a finalidade de se tratar os problemas que possam eventualmente aparecer.

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. 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.

O desenvolvimento de uma solução para auxiliar as empresas na correta configuração do SGBD de maneira automática, e também resolvendo questões de otimização e proteção, poderia oportunizar ganhos substanciais em relação ao ambiente computacional que hoje se conhece. A falta de conhecimentos adequados muitas vezes faz com que as empresas se preocupem mais com tecnologias de rede ou de hardware, pensando que com mais recursos computacionais os problemas se resolvem, quando uma correta configuração dos atuais recursos sanaria o problema.

O presente trabalho consiste no desenvolvimento de uma ferramenta capaz de efetuar as alterações necessárias para permitir que o SGBD funcione de maneira otimizada no que tange o aproveitamento dos recursos computacionais disponíveis no ambiente onde o mesmo está inserido. Possuindo uma base de conhecimento disponível pelo próprio SGBD, e utilizando uma série de entidades autonômicas, é capaz de se basear nos recursos do ambiente onde está inserido e interpretar algumas ações que ocorrem no SGBD, otimizando, configurando e protegendo o SGBD de maneira automática, sem que o usuário precise conhecer todos os parâmetros que serão alterados para que se aproxime do desempenho ideal. As técnicas utilizadas são detalhadas e, ao final, os resultados obtidos são apresentados.

No capítulo 2 estão detalhados os principais conceitos da computação autonômica. Este capítulo demonstra como os mesmos podem ser atingidos e quais os benefícios que se alcança ao implementá-los. Mostra os componentes básicos de um sistema autonômico, as tecnologias disponíveis e também fala da aplicação da computação autonômica para SGBDs. Em seguida, o capítulo 3 mostra alguns trabalhos correlatos, exemplificando o uso da computação autonômica em aplicações comuns e também o seu uso voltado para SGBDs. O capítulo 4 descreve todo o funcionamento da aplicação que foi construída para este projeto, mostrando seus componentes internos, a arquitetura do sistema e as ferramentas utilizadas. No capítulo 5 estão alguns experimentos que foram realizados em um SGBD enfatizando os resultados obtidos, e finalmente o capítulo 6 mostra as conclusões às quais se chegou a partir deste estudo.

2 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).

2.1 Características essenciais da computação autonômica

Para que se contorne todos estes desafios que estão aparecendo, a computação autonômica surge com uma série de conceitos que devem 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 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.

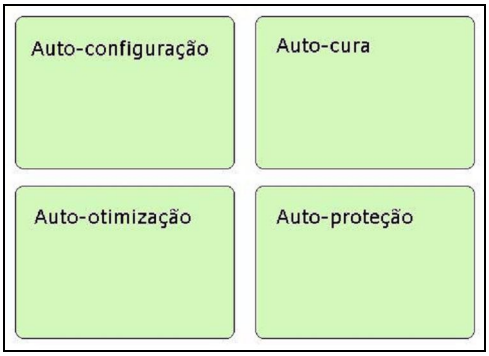

Figura 2: As Características Básicas de um Sistema Autonômico
Fonte: Autor.

2.1.1 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 deverão poder ser efetuadas imediatamente. (MANOEL *et al*., 2005).

2.1.2 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 que fique melhorando 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.

2.1.3 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.

2.1.4 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).

2.2 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 3 demonstra o laço de controle autonômico e a maneira como os componentes trabalham e comunicam-se entre si.

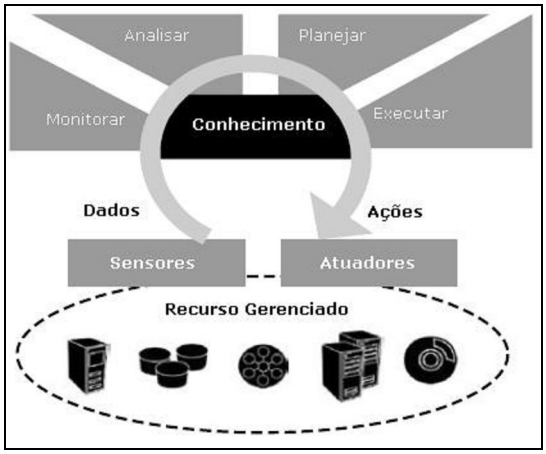

Figura 3: O Laço de Controle Autonômico
Fonte: PARASHAR e HARIRI (2007).

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 3, 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 formam 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.

2.3 O uso de computação autonômica para SGBDs

Os sistemas gerenciadores de bancos de dados (SGBDs) são aplicações antigas, 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 solicitado.

> 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).

Hoje em dia, 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 contrasta com o que 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).

Existem vários bons SGBD disponíveis hoje no mercado. Alguns são gratuitos, outros não. Entre os SGBDs gratuitos mais conhecidos estão o PostgreSQL, o Firebird e o Mysql. O PostgreSQL é o que fornece mais recursos ao usuário. Além disso, suporta um enorme volume de dados e possui uma velocidade de processamento bastante interessante. Já o Firebird é bastante difundido no meio comercial, e muitas pequenas empresas o adotam pela simplicidade de instalação, configuração e gerenciamento.

Apesar de todo o poder de processamento de um banco de dados, a atual exigência de performance 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.

Neste ponto a computação autonômica vem de encontro a esta necessidade crescente de uma performance otimizada. Muitas empresas vêem nela uma solução para que os SGBDs alcancem um patamar superior ao hoje conhecido, firmando-se como aplicações inteligentes e capazes de, através dos conceitos das características de auto-gerenciamento, otimizar-se sozinhos de modo que uma performance cada vez maior seja oferecida ao usuário sem que o mesmo precise ter o conhecimento técnico necessário para otimizar o sistema manualmente.

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 (SHASHA E BONNET, 2003). 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 mundos 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).

2.4 Sintonia do SGBD PostgreSQL

Pela riqueza de recursos e conformidade com os padrões, o PostgreSQL é um SGBD adequado para estudo universitário do modelo relacional, e uma ótima opção para empresas implementarem soluções de alta confiabilidade sem altos custos de licenciamento (POSTGRESQL-BR, 2008).

A correta configuração 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. Para outros, isto não é necessário.

O Quadro 1 demonstra alguns parâmetros de configuração do PostgreSQL (arquivo postgresql.conf), que possuem relação direta com o desempenho do SGBD, ou seja, provêem ao SGBD um melhor desempenho quando configurados da maneira correta.

Item	Descrição
checkpoint_segments	Número máximo de segementos 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 seta a quantidade de memória que o servidor do banco de dados utilize 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 (OTIMIZANDO BANCOS POSTGRESQL – PARTE 1, 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 (OTIMIZANDO BANCOS POSTGRESQL – PARTE 1, 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 1: Relação de Parâmetros de Configuração do PostgreSQL
Fonte: Manual do PostgreSQL (2008).

3 TRABALHOS CORRELATOS

Como resultado da atenção que tem sido dada aos SGBDs por parte da computação autonômica, alguns protótipos foram construídos, agentes implementados e melhorias propostas no sentido de se aplicar as técnicas de computação autonômica nos SGBDs que hoje se conhece. Este capítulo descreve em detalhes algumas implementações já realizadas aplicando-se computação autonômica a SGBDs.

3.1 LEO – *Learning Optmizer*

Como já foi dito anteriormente, a IBM tem um papel fundamental no ramo da computação autonômica. Foi através dela que tudo começou, e a empresa procura promover e incentivar o uso de práticas autonômicas ao redor do mundo.

No ano de 2002, a empresa desenvolveu um trabalho pioneiro, quando criou o LEO (*LEarning Optmizer*), um protótipo que era capaz de melhorar o desempenho de consultas no SGBD DB2 em determinadas ocasiões, e sob determinadas situações (MARKL, 2003). O LEO é apenas um protótipo que mostra em um item falho de um grande SGBD as vantagens de se utilizar o conceito de auto-otimização da computação autonômica.

De acordo com Markl (2003), o 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 poderá se beneficiar desta otimização.

> Um otimizador de consultas utiliza um modelo matemático de execução de consultas para determinar automaticamente a melhor maneira para acessar e processar qualquer consulta SQL data. Este modelo é bastante dependente das estimativas do otimizador a respeito do número de linhas que serão o resultado a cada passo do plano de execução de consultas, especialmente para consultas complexas envolvendo muitos predicados e/ou operações. Estas estimativas se baseiam em estatísticas no banco de dados que podem ou não ser verdadeiras. Monitorando as consultas a medida em que elas executam, o otimizador autonômico compara as estimativas do otimizador com as cardinalidades atuais, e computa ajustes nas suas estimativas que podem ser utilizados durante futuras otimizações de consultas similares (MARKL, 2003, p. 1).

A Figura 4 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.

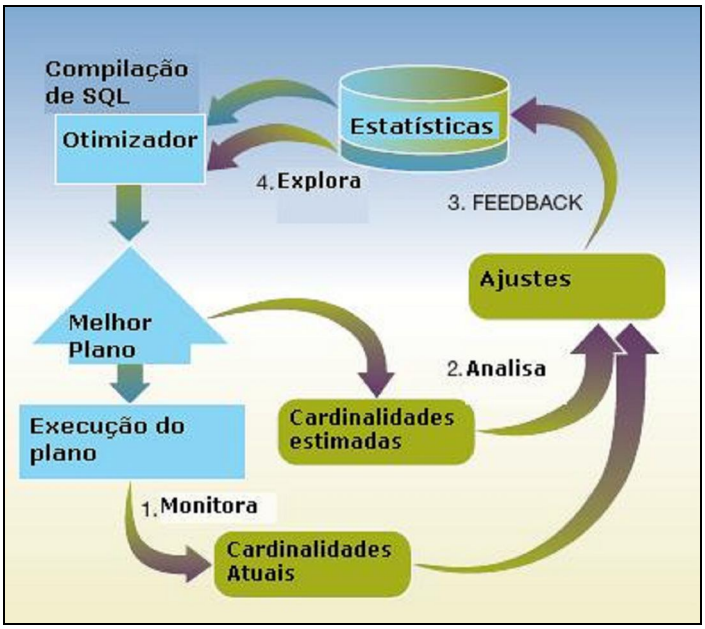

Figura 4: O Laço de *Feedback* do LEO
Fonte: MARKL (2003).

Conforme pode ser observado na Figura 4, a arquitetura do LEO é construída em um laço composto de quatro passos: monitorar, analisar, feedback e exploração do feedback.

3.2 Um agente de *software* para a construção de índices no PostgreSQL

De acordo com Entendendo e Usando Índices (2008), grandes melhorias no desempenho de um SGDB são atingidas no momento em que se passa a utilizar índices. Os índices aceleram a recuperação dos dados, pois funcionam tal qual o índice de um livro: não é necessário que se leia o livro página a página até que se encontre o tópico desejado, pode-se consultar o índice e dessa maneira é possível ir direto ao tópico em questão.

Como se pode perceber, a correta configuração de índices em um SGBD faz com que a performance do mesmo seja aumentada sempre que consultas ou outras operações fazem o uso do referido índice.

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. De acordo com Salles e Lifschitz (2004), 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.

> 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 (SALLES e LIFSCHITZ, 2004, p. 1).

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 (SALLES e LIFSCHITZ, 2004).

3.3 O uso da computação autonômica pelos SGBDs atuais

Um trabalho muito interessante foi realizado por Elnaffar *et al.* (2003), onde se verificou até que ponto os SGBDs atuais estão autonômicos. As características autonômicas foram pesquisadas, exploradas, e os resultados mostram o ponto até onde os SGBDs atuais tomam proveito dos benefícios da computação autonômica. 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.

4 ADA – UM AGENTE AUTONÔMICO PARA AUTO-SINTONIA EM BANCO DE DADOS POSTGRESQL

Após as pesquisas realizadas com os parâmetros do SGBD PostgreSQL que possuem relação com o seu desempenho, iniciou-se a construção da ferramenta ADA (Autonomic Database Agent).

De acordo com Russell e Norvig (2004), um agente é tudo o que pode ser considerado capaz de perceber seu ambiente por meio de sensores e de agir sobre esse ambiente por intermédio de atuadores.

Sendo assim, a ferramenta ADA é um agente autonômico capaz de, gradualmente, sintonizar os parâmetros do SGBD, 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 o seu desempenho.

4.1 Descrição geral

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.

4.1.1 Ferramentas utilizadas no desenvolvimento

O agente foi implementado em Java e utiliza a tecnologia JDBC para acesso ao banco de dados que será o alvo da otimização.

Java é uma linguagem moderna, poderosa e fácil de ser aprendida. (TROTTIER e WILLIANS, 2002). Há versões de Java para desenvolvimento de aplicações *desktop*, para desenvolvimento de aplicações para *internet*, entre outros. O Java foi anunciado formalmente

pela Sun em 1995 (DEITEL, 2003) e desde então vem sendo utilizada amplamente por programadores ao redor do mundo inteiro.

Entre as principais vantagens da linguagem Java estão a portabilidade das aplicações, a robustez e a facilidade de manutenção dos sistemas nela desenvolvidos, vantagem esta proveniente dos conceitos de orientação a objetos.

Um programa em Java pode ser definido como uma coleção de objetos que trocam mensagens entre si, conversando e invocando métodos uns dos outros. Cada objeto possui um tipo, e este tipo é definido através de uma classe ou interface (SIERRA e BATES, 2006).

De acordo com JDBC Overview (2008), o uso da tecnologia JDBC fornece uma série de vantagens em termos de usabilidade e desempenho no acesso ao SGBD. Entre as principais vantagens estão a forte integração entre a *Application Program Interface* (API) Java e a API JDBC, resultando em um desenvolvimento de aplicações fácil e econômico, e a facilidade de aprendizado e manutenção da tecnologia. Além disso, a tecnologia JDBC provê acesso completo aos metadados disponíveis no SGBD, e isto se torna interessante à medida em que a aplicação pode ter acesso a estes metadados para aumentar a sua base de conhecimento e descobrir algumas características do SGBD.

Para o desenvolvimento do agente, foi utilizada a IDE (*Integrated Development Environment*) NetBeans, versão 6.1.

> O NetBeans é um projeto *open source* de sucesso, com uma grande base de utilizadores, uma crescente comunidade e perto de 100 parceiros mundiais. A Sun Microsystems fundou o projeto NetBeans em junho de 2000 e continua a ser o seu principal patrocinador. Atualmente existem dois produtos: o IDE NetBeans (NetBeans IDE) e a Plataforma NetBeans (NetBeans Platform). O NetBeans IDE é um ambiente de desenvolvimento - uma ferramenta para programadores, que permite escrever, compilar, depurar e instalar programas. O IDE é completamente escrito em Java, mas pode suportar qualquer linguagem de programação. Existe também um grande número de módulos para extender as funcionalidades do IDE NetBeans. O NetBeans IDE é um produto livre, sem restrições à sua forma de utilização (NETBEANS.ORG, 2008, p. 1).

A Figura 5 demonstra o uso da IDE Netbeans, versão 6.1, que foi utilizada para o desenvolvimento do agente ADA.

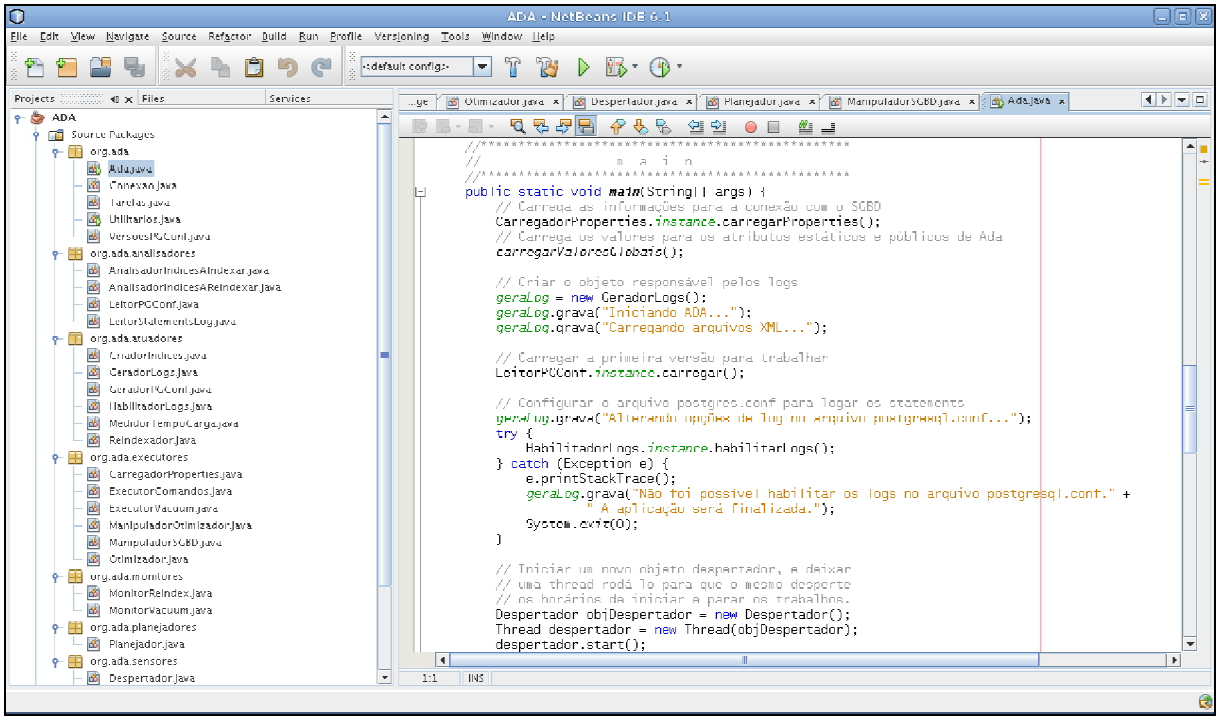

Figura 5: A IDE NetBeans
Fonte: Autor.

4.1.2 Funcionamento do agente

Após ser iniciado, o agente habilita os logs do SGBD que permitem que a carga real possa ser recuperada para depois ser executada em um ambiente de testes. Com os logs habilitados, o agente aguarda um horário pré-definido para que possa rodar as suas operações diárias, sendo que este horário pode ser definido pelo usuário. Inicialmente, o agente roda em um determinado período ao qual foi chamado de período de janela. Este período de janela é necessário para que o mesmo possa aguardar que se tenha uma quantidade suficiente de transações para realmente saber se o desempenho está melhorando ou piorando com as configurações definidas.

Com a finalidade de descobrir o desempenho do SGBD, o conjunto de transações que devem formar a carga é composto por operações de inserção, deleção, consulta e atualização nos dados do SGBD. Para que o SGBD passe a gravar em log as referidas operações, alguns

itens do arquivo postgresql.conf foram configurados de modo que o SGBD execute a tarefa desejada. O Quadro 2 demonstra o nome do parâmetro configurado, e o valor utilizado pelo agente para o parâmetro em questão.

Parâmetro	Descrição	Valor configurado pelo agente
log_destination	Método que será utilizado para gravar em arquivos de log as mensagens do servidor.	'stderr'
Logging_collector	Permite que as mensagens enviadas para stderr sejam capturadas e redirecionadas para arquivos de log (PostgreSQL 8.3: *Error Reporting and Logging*, 2008).	on
log_filename	Quando o parâmetro *logging_collector* está habilitado, o SGBD configura o nome do arquivo de log que será criado (PostgreSQL 8.3: Error Reporting and Logging, 2008). Podem ser utilizados caracteres de escape para que o SGBD grave no nome do arquivo informações como data e horário. O agente ADA utiliza as expressões %y (ano), %m (mês) e %d (dia). Dessa forma o agente sabe onde encontrar o conjunto de transações que foi executado no dia desejado.	'ada.%y.%m.%d.log'
log_statement	Controla quais instruções SQL serão gravadas no arquivo de log. Somente as instruções desejadas (instruções *select*, *insert*, *update* e *delete*) serão utilizadas poseteriormente.	all

Quadro 2: Relação dos Itens que são Configurados pelo Agente no Arquivo postgresql.conf
Fonte: Autor.

Os principais itens que se referem ao desempenho do PostgreSQL, e que serão trabalhados pelo agente, já foram definidos no capítulo 2.4, porém o usuário pode definir

novos itens a serem trabalhado, caso desejar. Sendo assim, todos os itens que serão trabalhados no arquivo postgresql.conf podem ser definidos pelo usuário, através de um arquivo xml criado para esta finalidade. Para poder trabalhar com os itens que possuem relação com o desempenho, o agente gera diferentes configurações de valores para os mesmos em arquivos postgresql.conf, e testa o desempenho do SGBD com cada arquivo gerado. Para testar o desempenho do SGBD, o agente submete a carga real, que foi submetida ao banco durante o período de janela, sobre uma cópia dos dados criada no início do período da janela. O agente então verifica o número de transações por segundo alcançado com as configurações atuais. Após isso, continua executando os testes de desempenho efetuando pequenas alterações no arquivo de configurações enquanto estiver obtendo uma performance superior à encontrada com as configurações anteriores. Terminado este trabalho diário, o agente restaura os dados reais do SGBD em questão e fica aguardando ser ativado novamente no dia seguinte.

O agente trabalha com versionamento do arquivo de configurações do PostgreSQL. Todos os conjuntos de configurações dos parâmetros, bem como os valores de cada um desses parâmetros em dado momento e o número de transações por segundo alcançado com cada um dos conjuntos é mantido em um vetor de versões do arquivo postgresql.conf. Dessa maneira, a qualquer momento é possível gerar o arquivo referente a qualquer estado pelo qual a aplicação já tenha passado em momentos anteriores.

No capítulo seguinte serão explicadas mais detalhadamente estas fases, bem como a arquitetura definida para esta aplicação.

4.2 Arquitetura e componentes da aplicação

Para a construção da aplicação foi utilizado o laço de controle autonômico. Conforme explicado anteriormente, o laço de controle autonômico é uma arquitetura bastante utilizada na área de computação autonômica, na qual os componentes (sensores, atuadores, monitores, analisadores, planejadores, executores e atuadores) são ligados a um recurso gerenciado e agem sobre e de acordo com um conhecimento específico.

A Figura 6 demonstra os componentes do ADA nesta arquitetura.

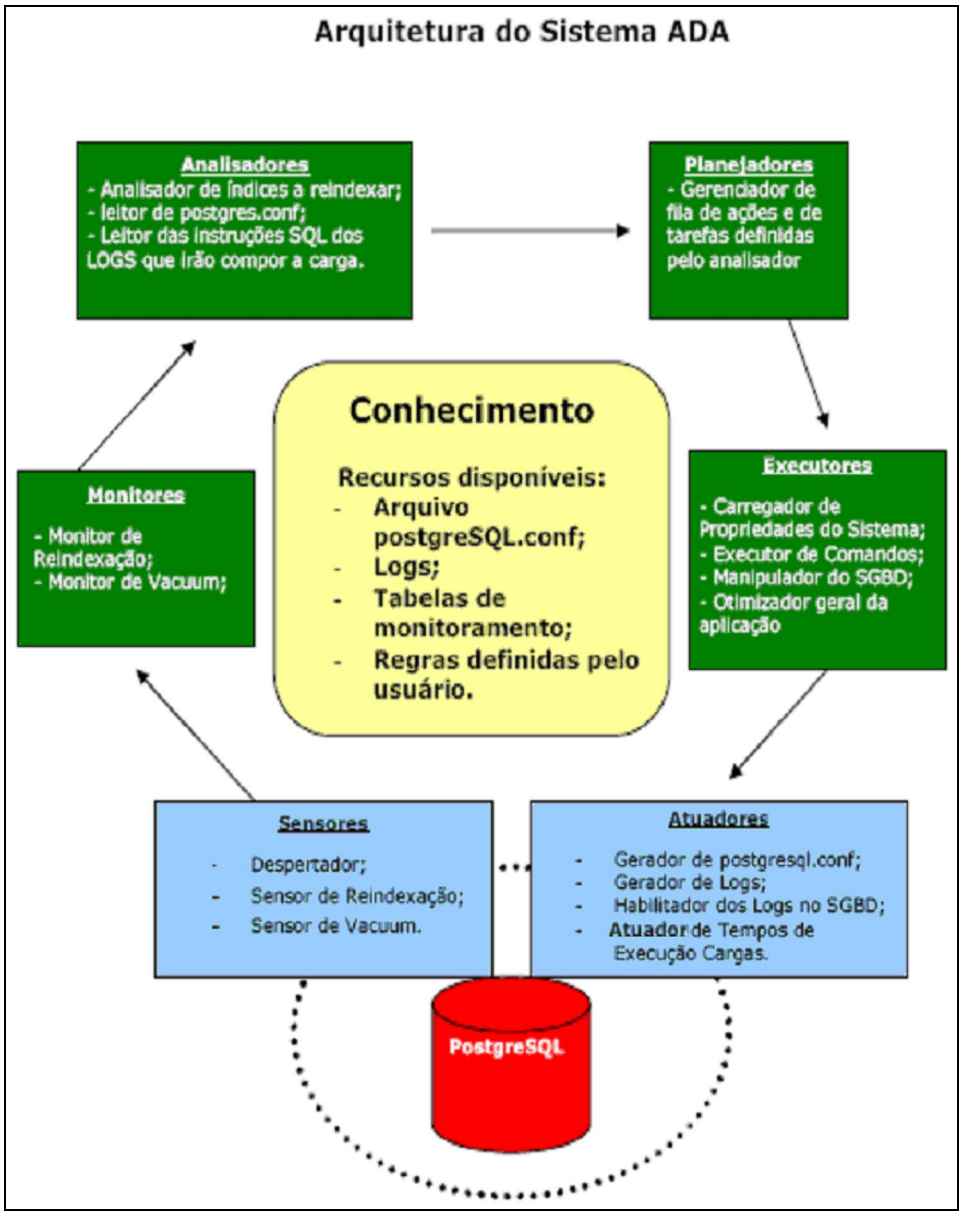

Figura 6: Arquitetura do Sistema ADA
Fonte: Autor.

Os quadros a seguir detalham os componentes do sistema, exibidos na Figura 2. O Quadro 3 descreve os sensores utilizados no agente e a funcionalidade de cada um deles.

Despertador	Como o agente permanece o tempo todo ligado ao SGBD, o mesmo possui um mecanismo despertador que ativa e desativa o seu funcionamento. Caso a hora diária de início das operações

	seja atingida, o despertador é o sensor que detectará este horário. O mesmo acontece para encerrar as atividades.
Sensor de Reindexação	De acordo com um tempo estabelecido previamente pelo usuário, o agente executa uma reindexação de alguns índices no SGBD. Este procedimento é feito apenas no momento correto, e é o sensor de reindexação o responsável por saber a chegada deste momento.
Sensor de *Vacuum*	De acordo com um tempo estabelecido previamente pelo usuário, o agente executa o procedimento de *Garbage Collection* próprio do PostgreSQL. Este procedimento é feito apenas no momento correto, e é o sensor de *vacuum* o responsável por saber a chegada deste momento.

Quadro 3: Relação de Sensores do Agente e o seu Funcionamento
Fonte: Autor.

O Quadro 4 descreve os monitores do agente e a finalidade de cada um.

Monitor de Reindexação	É o monitor responsável pela criação das tarefas de reindexação nos períodos disparados pelo sensor de reindexação. Este componente monitora o referido sensor e cria a tarefa de reindexação, a qual será posteriormente executada pelo componente responsável.
Monitor de *Vacuum*	É o monitor responsável pela criação das tarefas de *Vacuum* nos períodos disparados pelo SensorVacuum. Este componente monitora o referido sensor e cria a tarefa de reindexação, a qual será posteriormente executada pelo componente responsável.

Quadro 4: Relação de Monitores do Agente e o seu Funcionamento
Fonte: Autor.

O Quadro 5 descreve os analisadores do agente e a funcionalidade de cada um.

Analisador de Índices a reindexar	Este analisador é o responsável por analisar quais são os índices que devem sofrer uma reindexação no próximo momento em que o monitor criar a tarefa de reindexação, e o planejador a colocar na fila de ações a serem executadas.
Leitor de postgresql.conf	É o responsável pela análise completa do arquivo de configurações do PostgreSQL, o postgresql.conf. Este analisador verifica este arquivo, carregando as propriedades que cada um dos itens possui originalmente para que o agente possa começar a modificá-las.
Leitor de Instruções SQL dos *Logs*	Este analisador é o responsável pela análise detalhada dos *logs* do SGBD, à procura das instruções que foram enviadas ao SGDD durante o período da janela que está sendo trabalhada. Após a análise do arquivo e verificação dos itens, este analisador carrega os valores para as suas respectivas estruturas internas, e estas instruções serão guardadas para que possam ser simuladas posteriormente.

Quadro 5: Relação de Analisadores do Agente e o seu Funcionamento
Fonte: Autor.

O Quadro 6 descreve as funcionalidades do planejador do agente ADA.

Planejador (Gerenciador da Fila de Ações)	O agente possui um componente central responsável pelo planejamento. Este planejador verifica de tempos em tempos as tarefas que foram criadas pelos outros componentes da aplicação, checa a prioridade de cada tarefa e decide qual tarefa deve ser executada quando, disparando as tarefas com as maiores prioridades e mantendo as tarefas menos importantes em uma fila de espera a qual ele próprio gerencia.

Quadro 6: Finalidade do Planejador de Aplicação
Fonte: Autor.

O Quadro 7 descreve os executores do agente e funcionalidade de cada um.

Carregador de Propriedades do Sistema	Este executor é o responsável pelo carregamento das configurações gerais do agente, ou seja, as regras de alto-nível definidas pelo usuário. Ao usuário é permitido definir qual o SGBD, o horário de funcionamento do agente, os itens que serão trabalhados, a escala de alteração dos valores de cada um, os limites mínimo e máximo que devem ser observados ao se calcular um novo valor para o item, entre outros.
Executor de Comandos	É o responsável pela execução de comandos contendo instruções SQL no SGBD. Estes comandos encontram-se em tarefas que foram criadas por outros componentes. Além de comandos comuns a um SGBD, este executor também executa o v*acuum* e a reindexação de índices.
Manipulador do SGBD	Este executor é responsável pela manutenção de operações que são executadas para a administração do SGBD. Em geral, o mesmo se encarrega de iniciar o serviço do banco, encerrá-lo, e criar espelhamentos de dados.
Otimizador	Este executor é um dos componentes mais importantes da aplicação. A sua função é a de garantir a seqüência dos testes de desempenho realizados em cada iteração, e a de manter o próprio laço de testes em funcionamento enquanto estiver no período permitido pelo usuário. Ele solicita ao componente responsável a execução da carga carregada, organiza os tempos de cada um dos experimentos para cada mudança de valor dos itens trabalhados e ainda verifica se deve parar ou então avançar nos testes diários.

Quadro 7: Relação de Executores do Agente e o seu Funcionamento
Fonte: Autor.

O Quadro 8 descreve os atuadores do agente e a finalidade de cada um.

Gerador de *Logs*	Este atuador é o responsável por atender as solicitações de geração de *logs* no arquivo de *logs* da aplicação.
Gerador do postgresql.conf	Este atuador recria fisicamente o arquivo de configurações do PostgreSQL, gerando os valores de qualquer versão de arquivo solicitada por qualquer outro componente.
Habilitador de *Logs*	Este atuador é utilizado no momento em que a aplicação inicia. Sua tarefa principal é ativar os *logs* do PostgreSQL corretamente para que o SGBD passe a gravar em *log* as operações de *insert*, *select*, *update* e *delete* realizadas. Desta maneira, estas instruções poderão posteriormente ser recuperadas.
Atuador de tempos de cargas	Este atuador é o responsável por executar instruções SQL no SGBD a pedidos do otimizador, e também de medir o tempo dispensado para a execução de um bloco de instruções compreendidos em um período de janela qualquer.

Quadro 8: Relação de Atuadores do Agente e o seu Funcionamento
Fonte: Autor.

4.3 Arquivos de configuração do agente

Uma vez iniciado, o agente carrega alguns arquivos XML que contém as configurações gerais que serão utilizadas pelo mesmo. Este arquivo mantém as regras definidas pelo usuário, geralmente um DBA. Através destes arquivos, uma grande variedade de opções está disponível para a configuração do agente pelo usuário.

4.3.1 Arquivo config.xml

Este é o arquivo onde o usuário pode realizar as configurações gerais do agente. Neste arquivo estão definidos os tempos de início e término dos trabalhos diários, o intervalo para que o *Vacuum* e a re-criação de índices sejam executados, e também o nome do arquivo que será utilizado para gerar os logs do agente, o qual conterá detalhes de todas as operações efetuadas pelo agente, informando ao usuário as operações que foram executadas e as ações que foram tomadas.

Através do Quadro 9 apresentado, pode-se visualizar as configurações gerais que podem ser feitas para o agente.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE properties SYSTEM "http://java.sun.com/dtd/properties.dtd">
<properties>
   <comment>Opcoes gerais da aplicacao</comment>
   <entry key="tempoPadrao">60</entry>
   <entry key="horaInicio">10</entry>
   <entry key="horaFim">14</entry>
   <entry key="intervaloVacuum">180</entry>
   <entry key="intervaloReindex">10</entry>
   <entry key="arquivoLog">ada.log</entry>
   <entry key="pastaEspelho">dataEspelho</entry>
   <entry key="janela">2</entry>
</properties>
```

Quadro 9: Conteúdo do Arquivo config.xml
Fonte: Autor.

O Quadro 10 descreve os parâmetros que podem ser configurados e a finalidade de cada um.

Item	Finalidade
HoraInicio	O horário diário de início das atividades do agente.
HoraFim	O horário diário de término das atividades do agente.
intervaloVacuum	O intervalo de tempo, em minutos, para que seja executada a tarefa de Vacuum.
intervaloReindex	O intervalo de tempo, em minutos, para que seja executada a tarefa de Reindex.

arquivoLog	O nome do arquivo de log que deve ser gerado pelo agente.
pastaEspelho	O nome da pasta que será a cópia dos dados que serão trabalhados pelo agente. O nome da pasta definida aqui recebe ainda a identificação de dia, mês e ano de criação da pasta, para que desta forma o agente possa identificar qual a pasta que deve ser utilizada a cada dia.
Janela	O número de dias que o agente deverá utilizar como período de janela.

Quadro 10: Itens Passíveis de Configuração no Arquivo config.xml
Fonte: Autor.

4.3.2 Arquivo configLog.xml

Este arquivo contém o conjunto de opções que devem ser utilizadas para o log do SGBD PostgreSQL. As configurações definidas neste arquivo serão utilizadas para habilitar a gravação dos comandos que são necessários para a execução do agente. Dessa maneira, caso seja necessário uma configuração diferente da maneira como o SGBD grava os logs, a mesma pode ser definida neste arquivo.

Através do Quadro 11 apresentado pode-se visualizar as configurações gerais que podem ser feitas para o agente.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE properties SYSTEM "http://java.sun.com/dtd/properties.dtd">
<properties>
<comment>Config do log</comment>
<entry key="log_filename">ada.%y.%m.%d.log</entry>
<entry key="logging_collector">on</entry>
<entry key="log_statement">all</entry>
</properties>
```

Quadro 11: Conteúdo do Arquivo configLog.xml
Fonte: Autor.

O Quadro 12 descreve a finalidade dos parâmetros que podem ser configurados.

Item	Finalidade
log_filename	O nome do arquivo de log que o SGBD deverá manter.
logging_collector	Habilitação da propriedade que permite que os comandos do SGBD sejam gravados em log.
log_statement	Informa o que deve ser logado para que o agente trabalhe posteriormente.

Quadro 12: Itens Passíveis de Configuração no Arquivo configLog.xml
Fonte: Autor.

4.3.3 Arquivo db.xml

Este arquivo contém as configurações do SGBD ao qual o agente irá se conectar para realizar os trabalhos. No arquivo são definidos o nome da máquina servidora do banco de dados, o usuário e senha do banco de dados, a base de dados que será utilizada, entre outros.

4.3.4 Arquivo items.xml

Neste arquivo são definidos os itens que serão trabalhados pelo agente. Os itens que afetam a performance do SGBD já estão previamente configurados, porém o usuário pode livremente adicionar novos itens para serem trabalhados, e desta maneira o agente poderá verificar se mudanças nos valores deste novo parâmetro afetarão o desempenho do SGBD.

Para cada item do SGBD um valor correspondente é configurado e chamado de escala de mudança (delta), ou seja, o total da mudança que cada item irá sofrer para que se possa executar o próximo teste com o valor alterado. A definição dos itens bem como a escala de alteração de cada um pode ser observada no Quadro 13.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE properties SYSTEM "http://java.sun.com/dtd/properties.dtd">
<properties>
<comment>Itens/escalas a serem trabalhados</comment>
<entry key="checkpoint_segments">8</entry>
<entry key="checkpoint_timeout">2</entry>
<entry key="shared_buffers">8</entry>
<entry key="effective_cache_size">32</entry>
<entry key="wal_buffers">32</entry>
<entry key="commit_delay">1000</entry>
</properties>
```

Quadro 13: Conteúdo do Arquivo itens.xml
Fonte: Autor.

4.3.5 Arquivo limites.xml

Neste arquivo são definidos os intervalos máximos permitidos para cada item que será trabalhado. O agente estima valores para cada nova iteração através dos itens, e considera os limites estabelecidos neste arquivo para não atravessar nenhuma barreira que pode ser definida pelo usuário.

Sendo assim, cada item possui um intervalo de possíveis valores que podem ser utilizados pelo agente. Esta definição pode ser observada no Quadro 14.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE properties SYSTEM "http://java.sun.com/dtd/properties.dtd">
<properties>
<comment>Limites dos itens trabalhados</comment>
<entry key="checkpoint_segments">1-30</entry>
<entry key="checkpoint_timeout">2-30</entry>
<entry key="shared_buffers">16-512</entry>
<entry key="effective_cache_size">16-1024</entry>
<entry key="wal_buffers">32-512</entry>
<entry key="commit_delay">0-5000</entry>
</properties>
```

Quadro 14: Conteúdo do Arquivo limites.xml
Fonte: Autor.

4.4 Fluxo de trabalho do agente

Através da Figura 7 é possível um melhor entendimento do modo que o agente trabalha. Após o início das atividades, o planejador principal da aplicação, responsável por

disparar as tarefas certas nos momentos certos, permanece ativo e iniciando uma série de procedimentos que fazem parte dos testes necessários para otimizar o desempenho do SGBD.

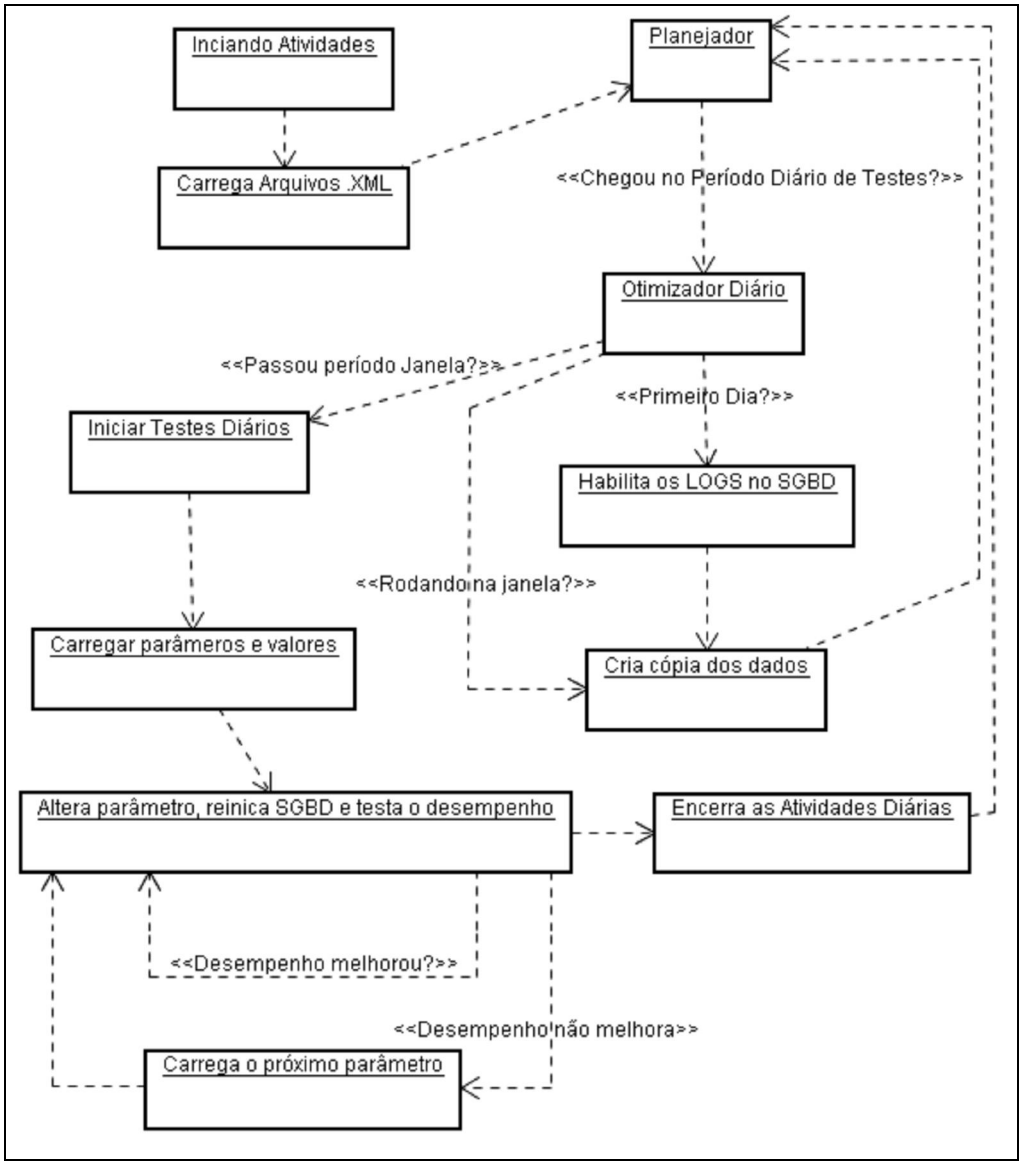

Figura 7: Fluxo de Trabalho do Sistema ADA
Fonte: Autor.

Após o planejador definir as tarefas que serão executadas, o otimizador controla todos os testes e verifica o desempenho obtido com as várias configurações utilizadas nos parâmetros.

Todos os testes são realizados sobre os dados disponíveis no SGBD no momento anterior ao período de janela definido. Dessa forma, operações de inserção e deleção podem ser efetuadas normalmente sobre os dados, já que os mesmos caracterizam um ambiente paralelo com a finalidade exclusiva de servir para a verificação realizada pelo agente.

Após um período de janela definido previamente pelo usuário, o agente está pronto para iniciar os testes. Para isto, o mesmo permanece mudando os valores dos itens e registrando as melhores no desempenho atingidas com estas mudanças. O agente então finaliza as atividades diárias, seja pelo fato de ter atingido o tempo limite de testes diários permitidos pelo usuário, seja pelo fato de ter encontrado o melhor desempenho com os parâmetros trabalhados para a carga de operações que foi utilizada.

Para encontrar os melhores valores para cada parâmetro, existe um algoritmo utilizado pelo agente. Ao iniciar os trabalhos com determinado parâmetro, ADA carrega o seu valor padrão, e então começa a decrementar este valor utilizando a escala definida pelo usuário. Por exemplo, se o valor original para determinado parâmetro é de 32kb, e a escala é de 4kb, o próximo valor calculado será 28kb. Após este cálculo, o agente inicia os testes com o valor deste parâmetro alterado. Caso o desempenho tenha melhorado, o agente continua decrementando o valor e verificando o novo desempenho obtido. Caso contrário, o agente incrementa o valor do parâmetro e efetua os mesmos testes para verificação do desempenho, porém agora sempre incrementando o valor do parâmetro enquanto um desempenho superior ao anterior estiver sendo encontrado. Tão logo um desempenho inferior seja encontrado, ou então o mesmo desempenho tenha se mantido por três testes subseqüentes, o agente pára os testes diários, encerra as suas atividades e permanece aguardando pelo próximo dia de trabalho.

Após a finalização dos testes diários, o agente remove os dados que foram utilizados para que se executasse os testes, dados estes referentes ao dia anterior ao período de janela disponível. Esta deleção é permitida, pois os referidos dados foram utilizados somente para os testes do dia em questão. Para o dia seguinte uma nova pasta, contendo os dados do dia subseqüente ao utilizado no dia anterior, estará disponível e servirá de ambiente de testes.

5 EXPERIMENTOS

Este capítulo descreve em detalhes a execução do agente, bem como os resultados que foram obtidos a partir de uma execução do mesmo. Também demonstra os passos a serem executados com a finalidade de se utilizar a aplicação.

5.1 Etapas para iniciar a execução do agente

A primeira atividade a ser realizada é a configuração desejada dos parâmetros disponíveis nos arquivos XML, conforme demonstrado no capítulo 4.3. Esta configuração deve ser realizada sempre que o usuário desejar modificar os valores padrão que o agente utilizará. Após esta fase de configuração, que tem caráter opcional, o agente já pode ser iniciado.

Por ser uma aplicação desenvolvida em Java, o arquivo compilado da aplicação compõe um arquivo de pacote java chamado ada.jar. Para iniciar a execução do agente, deve-se utilizar o comando mostrado na Figura 8, após estarmos localizados no diretório que contém o arquivo ada.jar.

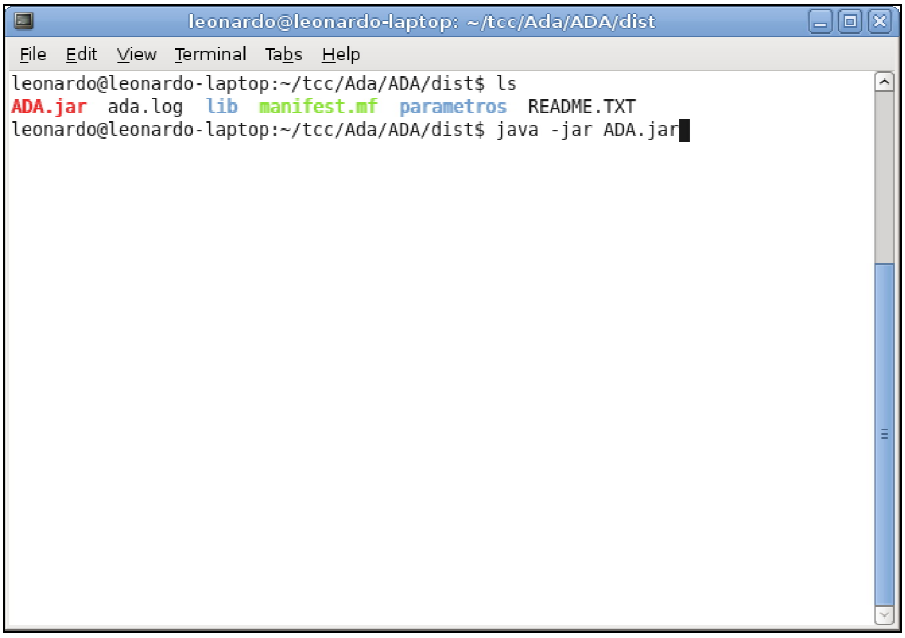

Figura 8: Inicializando o Agente
Fonte: Autor.

5.2 As Mensagens enviadas para a tela

Como o agente utiliza a saída do sistema para gerar as mensagens do que está sendo executado, a própria tela de *prompt* do sistema operacional exibe informações acerca das operações que estão sendo executadas no momento. A aplicação gera como saída linhas contendo a data corrente da operação que foi executada, o horário em que a mesma ocorreu e após isto informa a atividade que foi realizada. A tela de saída com as mensagens geradas pelo agente pode ser observada da Figura 9.

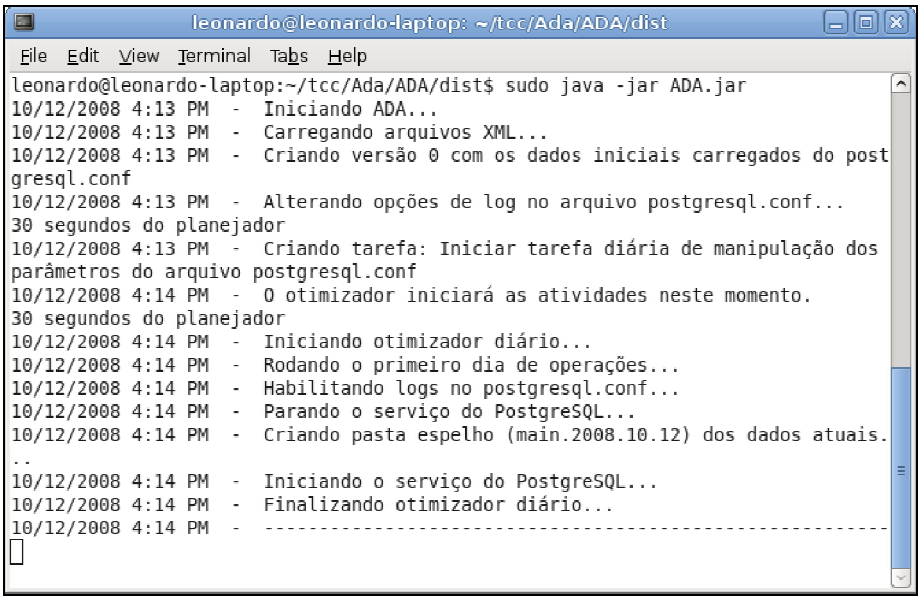

Figura 9: A Tela com as Saídas do Sistema
Fonte: Autor.

5.3 O Arquivo de logs gerado pelo agente

Ao mesmo tempo em que o agente gera as saídas na tela, o mesmo mantém um arquivo de log que pode ser consultado posteriormente pelo usuário a fim de que não seja necessário o acompanhamento on-line por parte de um DBA com a finalidade de que se tenha conhecimento de todas as operações que o agente está executando. Desta maneira, basta observar-se o arquivo de log gerado pelo agente para que se saiba exatamente as operações que foram executadas, bem como a data e hora de cada uma delas.

O nome do arquivo de logs gerado pelo agente previamente é configurado para ada.log, porém este nome pode ser alterado nos arquivos de configuração demonstrados no capítulo 4.3.

O apêndice A demonstra parcialmente o conteúdo do arquivo de logs gerado pelo agente.

5.4 A carga utilizada nos experimentos

A carga utilizada consistiu em algumas operações de inserção, deleção, alteração e exclusão de dados no SGBD. Não foi utilizada uma carga que pudesse ser caracterizada por qualquer definição desenvolvida pelo TPC (*Transaction Processing Performance Council*).

De acordo com Rstoever.com (2008), o TPC-C, por exemplo, é um *benchmark* de OLTP (processamento de transações *on-line*) desenvolvido pelo TPC. O *benchmark* TPC-C define um padrão rigoroso para calcular o desempenho em transações por minuto (tpmC).

A razão de não ter sido utilizada uma carga que pudesse ser caracterizada por um TPC reside no fato de que o agente, por ser autonômico, deve levar em conta as mudanças que ocorrerão freqüentemente na caracterização da carga que está sendo utilizada no SGBD. Utilizando um TPC estas mudanças não seriam percebidas e o desempenho poderia ser afetado por configurações errôneas para valores de parâmetros que seriam alterados em função de decisões que não condizem com o que realmente está acontecendo no SGBD.

5.5 Resultados

Com o objetivo de apurar-se os resultados que podem ser obtidos a partir da execução do agente, uma simulação foi executada com a carga descrita no capítulo anterior. Com os parâmetros definidos em suas configurações padrão, foi simulada a execução do agente em um período de 3 dias, onde observaram-se resultados significativos, como pode ser observado a seguir.

O arquivo de logs referente ao experimento realizado pode ser observado no capítulo 5.3. Neste arquivo estão demonstradas as operações que foram executadas, e pode-se observar claramente a melhora no desempenho do SGBD.

Como pode ser observado no log de execução demonstrado anteriormente, o agente inicia suas atividades alterando os valores dos parâmetros em uma forma gradual, de acordo co os incrementos especificados pelo usuário. Após alterar o valor de um parâmetro o agente então realiza os testes com a carga real submetida ao SGBD para que se obtenha o novo desempenho, apurado em TPS (transações por segundo), o qual demonstra o resultado das novas configurações que foram definidas. Se o desempenho melhorou, o agente modifica novamente o valor do parâmetro, e continua os testes até que o tempo limite para os trabalhos foi atingidos, ou o valor limite para o parâmetro em questão foi alcançado. Neste caso, o agente inicia os testes com o novo parâmetro disponível, iniciando novamente os testes de desempenho.

O Quadro 16 demonstra as alterações que foram feitas nos valores dos parâmetros durante o experimento, e os resultados obtidos, medidos em TPS.

Parâmetro	Valor Anterior	Novo Valor	TPS Anterior	Novo TPS
Checkpoint_segments	-	3	11,99	13,24
Checkpoint_segments	3	11	13,24	13,49
Shared_buffers	24	32	13,49	13,75
Effective_cache_size	96	128	13,75	14,00

Quadro 15: Resultados Obtidos pelo Agente
Fonte: Autor.

Como pode ser observado no Quadro 16, o agente trabalhou com vários parâmetros, porém os parâmetros demonstrados foram os que realmente ocasionaram um ganho de desempenho ao terem os seus valores alterados, para os testes com a carga fornecida.

Neste experimento, o valor do item *Checkpoint_segments* foi elevado até 11, e então as alterações neste item foram interrompidas, pois não se observou uma melhora no desempenho ao se utilizar um valor superior a 11 para o referido item.

Como pode ser visto, o agente obteve, neste caso, um ganho de performance de 16,76%. Uma questão importante que merece ser enfatizada é a dinâmica relação que existe entre os parâmetros que estão sendo sintonizados. A partir do momento em que o agente

altera o valor de um parâmetro, não há garantias de que a configuração dos outros parâmetros deve ser também otimizada de modo que a performance melhor mais ainda. Este é um ponto que o agente ADA sempre considera, pois no dia seguinte, quando os testes se iniciarem novamente, os valores alterados no dia anterior serão levados em conta para que se efetuem os testes com novos parâmetros. Deste modo, os valores de todos os parâmetros sempre são considerados ao se realizar um teste.

6 CONCLUSÃO

Apesar de ser uma área de pesquisa bastante recente, a computação autonômica está crescendo rapidamente. Grandes instituições como NASA, Sun, Cisco, HP e IBM estão explorando amplamente suas idéias, sendo que a última delas citada investiu dois bilhões de dólares em pesquisas na área (UNICAMP, 2008).

A computação autonômica pode ser aplicada a qualquer área da computação, sendo que os SGBDs, por seu uso disseminado e popular, carecem com urgência de soluções deste tipo. Além do alto custo de profissionais qualificados para a administração do banco de dados, esta é uma área onde os conceitos da computação autonômica mostram resultados claros e diretos.

O uso de agentes de software para implementar técnicas de computação autonômica em SGBDs tem se mostrado uma solução bastante eficiente quando problemas de otimização requerem um acurado conhecimento para que sejam solucionados.

Através da ferramenta ADA, ficou demonstrado que o uso de um agente de software para auto-sintonia de parâmetros no SGBD melhorou consideravelmente o desempenho do SGBD PostgreSQL. A partir dos resultados obtidos, após as mudanças efetuadas nas configurações dos itens relacionados ao desempenho do SGBD, pode-se notar claramente os benefícios que uma sintonia correta, que sempre considera o ambiente onde o SGBD está inserido, pode trazer em termos de ganhos de desempenho.

A ferramenta ADA é um agente que eficientemente melhora o desempenho do SGBD, sempre utilizando a carga real para determinar os melhores valores para os parâmetros configuráveis. Além do ganho de desempenho, a ferramenta ADA faz com que, após inicializada, a variação da carga de trabalho do SGBD não seja fonte de preocupação por parte do DBA na hora de configurar o SGBD, pois o dinamismo desta situação sempre é levado em conta.

Como trabalhos futuros, alguns pontos do agente podem ser melhorados. Um fator claro que deve ser verificado é o número de conexões simultâneas utilizadas para as simulações e testes. O agente ADA simula apenas um cliente conectado ao SGBD, e esta é uma necessidade real para que se chegue mais próximo ainda aos valores ideais para os parâmetros do SGBD PostgreSQL. Outro aspecto importante é a possibilidade de o agente descobrir por si próprio a hora certa de iniciar os trabalhos, utilizando recursos de auto-configuração para obter a otimização necessária no tempo certo. Embutir esta funcionalidade

ao agente o tornaria ainda mais autonômico.

Outro ponto que deve ser pesquisado em detalhes é a relação existente entre os diferentes tipos de carga e o melhor conjunto de valores de parâmetros para a referida carga. Através deste estudo podem ser descobertos os melhores conjuntos de valores através da caracterização da carga que está sendo utilizada.

REFERÊNCIAS

Autonomic computing. Disponível em: <http://www.research.ibm.com/autonomic/>. Acesso em: 28 ago. 2007.

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

DEITEL, H.M. **Java:** Como programar. Porto Alegre: Bookman, 2003.

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

Entendendo e Usando Índices. Disponível em: <www.devmedia.com.br/articles/viewcomp.asp?comp=6567> Acesso em: 03 nov. 2008.

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

PostgreSQL-BR. Disponível em: <http://www.postgresql.org.br/Introdu%C3%A7%C3%A3o_e_hist%C3%B3rico> Acesso em: 11 out. 2008.

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

JDBC OVERVIEW. Disponível em: < http://java.sun.com/products/jdbc/overview.html> Acesso em: 07 out. 2008

KEPHART, Jeffrey; CHESS, David. The vision of Autonomic Computing. **IEEE Computer Society**, p. 41-50, jan/2003.

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

Manual do PostgreSQL. Disponível em: <http://www.postgresql.org/docs/current/static/> Acesso em: 16 out. 2008

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

MURCH, Richard. **Autonomic computing**. Nova Iorque: IBM Press, 2004.

NAVEGA. Disponível em: <http://www.intelliwise.com/reports/info2001.htm> Acesso em: 07 nov. 2007

NETBEANS.ORG. Disponível em: < http://www.netbeans.org/index_pt_PT.html > Acesso em: 12 out. 2008

Otimizando Bancos PostgreSQL – Parte 01. Disponível em: <http://www.linuxdepot.com.br/otimizando_postg resql.txt> Acesso em: 11 jul. 2008.

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

PostgreSQL 8.3: Error Reporting and Logging. Disponível em: <http://www.postgresql.org/docs/current/static/runtime-config-logging.html > Acesso em: 07 out. 2008.

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

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

RSTOEVER.COM. Disponível em: < http://www.rstoever.com> Acesso em: 12 out. 2008

RUSSEL, S.; NORVIG, P. **Inteligência Artificial.** Rio de Janeiro: Elsevier, 2004.

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, out. 2004.

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

SIERRA, Kathy; BATES, Bert. **Certificação Sun Para Programador Java 5.** Rio de Janeiro: Alta Books, 2006.

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

TROTTIER, Alain; Willians, Al. **Java 2 core language.** Scottsdale: Paraglyph, 2002.

UNICAMP. **Computação Autonômica: Desafios e Oportunidades**. Disponível em: <http://www.ic.unicamp.br/cpg/interno/seminarios/pesquisa/31-agosto-2007-computacao-autonomica-desafios-oportunidades/>. Acesso em: 14 out. 2008.

APÊNDICE

APÊNDICE A - Log gerado pelo Agente

```
10/07/2008 12:32 AM  -  Iniciando ADA...
10/07/2008 12:32 AM  -  Carregando arquivos XML...
10/07/2008 12:32 AM  -  Criando versão 0 com os dados iniciais carregados do
postgresql.conf
10/07/2008 12:32 AM  -  Alterando opções de log no arquivo postgresql.conf...
10/07/2008 12:33 AM  -  Criando tarefa: Iniciar tarefa diária de manipulação dos
parmetros do arquivo postgresql.conf
10/07/2008 12:33 AM  -  O otimizador iniciará as atividades neste momento.
10/07/2008 12:33 AM  -  Iniciando otimizador diário...
10/07/2008 12:33 AM  -  Rodando o primeiro dia de operações...
10/07/2008 12:33 AM  -  Habilitando logs no postgresql.conf...
10/07/2008 12:33 AM  -  Parando o serviço do PostgreSQL...
10/07/2008 12:33 AM  -  Criando pasta espelho (main.2008.7.10) dos dados atuais...
11/07/2008 12:33 AM  -  Iniciando o serviço do PostgreSQL...
11/07/2008 12:33 AM  -  Finalizando otimizador diário...

...

12/07/2008 12:34 AM  -  Criando tarefa: Iniciar tarefa diária de manipulação dos
parmetros do arquivo postgresql.conf
12/07/2008 12:34 AM  -  O otimizador iniciará as atividades neste momento.
12/07/2008 12:34 AM  -  Iniciando otimizador diário...
12/07/2008 12:34 AM  -  Carregando os statements do arquivo de log do postgresql...
12/07/2008 12:34 AM  -  Lendo e parseando o arquivo de logs ada.08.07.10.log
12/07/2008 12:34 AM  -  Lendo e parseando o arquivo de logs ada.08.07.11.log
12/07/2008 12:34 AM  -  Parando o serviço do PostgreSQL...
12/07/2008 12:34 AM  -  Criando cópia da pasta de dados para manter backup
12/07/2008 12:34 AM  -  Restaurando a pasta de dados a partir do espelho
main.2008.7.10 para iniciar testes...
12/07/2008 12:34 AM  -  Iniciando o serviço do PostgreSQL...
12/07/2008 12:34 AM  -  Rodando os testes para encontrar a performance com a versão 0
(valores default carregados de postgresql.conf)...
12/07/2008 12:35 AM  -  1º teste - TPS Encontrado: 7.777777777777778
12/07/2008 12:35 AM  -  2º teste - TPS Encontrado: 12.727272727272727
12/07/2008 12:35 AM  -  3º teste - TPS Encontrado: 14.0
12/07/2008 12:35 AM  -  4º teste - TPS Encontrado: 12.727272727272727
12/07/2008 12:35 AM  -  5º teste - TPS Encontrado: 12.727272727272727
12/07/2008 12:35 AM  -  TPS médio encontrado: 11.991919191919191
12/07/2008 12:35 AM  -  Parando o serviço do PostgreSQL...
12/07/2008 12:36 AM  -  Restaurando a pasta de dados a partir do espelho
main.2008.7.10 para iniciar testes...
12/07/2008 12:36 AM  -  Iniciando o serviço do PostgreSQL...
12/07/2008 12:36 AM  -  Iniciando trabalhos com checkpoint_segments

...

12/07/2008 12:36 AM  -  Valor anterior encontrado: -5
12/07/2008 12:36 AM  -  Novo valor calculado: 3
12/07/2008 12:36 AM  -  Criando versão 1 do arquivo postgresql.conf
12/07/2008 12:36 AM  -  Parando o serviço do PostgreSQL...
12/07/2008 12:36 AM  -  Iniciando o serviço do PostgreSQL...
12/07/2008 12:36 AM  -  Iniciando testes com novo postgresql.conf nos dados,
utilizando postgresql.conf versão 1...
12/07/2008 12:37 AM  -  1º teste - TPS Encontrado: 12.727272727272727
12/07/2008 12:37 AM  -  2º teste - TPS Encontrado: 14.0
12/07/2008 12:37 AM  -  3º teste - TPS Encontrado: 12.727272727272727
12/07/2008 12:37 AM  -  4º teste - TPS Encontrado: 12.727272727272727
12/07/2008 12:37 AM  -  5º teste - TPS Encontrado: 14.0
12/07/2008 12:37 AM  -  TPS médio encontrado: 13.236363636363638
12/07/2008 12:37 AM  -  Tempo já decorrido nos trabalhos com checkpoint_segments: 1
minutos.
12/07/2008 12:37 AM  -  Tempo total permitido para trabalho neste item: 40 minutos.
12/07/2008 12:37 AM  -  Valor anterior encontrado: 3
12/07/2008 12:37 AM  -  Novo valor calculado: 11
12/07/2008 12:37 AM  -  Criando versão 2 do arquivo postgresql.conf
12/07/2008 12:37 AM  -  Parando o serviço do PostgreSQL...
12/07/2008 12:37 AM  -  Iniciando o serviço do PostgreSQL...
12/07/2008 12:37 AM  -  Iniciando testes com novo postgresql.conf nos dados,
utilizando postgresql.conf versão 2...
12/07/2008 12:38 AM  -  1º teste - TPS Encontrado: 12.727272727272727
12/07/2008 12:38 AM  -  2º teste - TPS Encontrado: 14.0
12/07/2008 12:38 AM  -  3º teste - TPS Encontrado: 14.0
```

```
12/07/2008 12:38 AM  -  4º teste - TPS Encontrado: 12.727272727272727
12/07/2008 12:38 AM  -  5º teste - TPS Encontrado: 14.0
12/07/2008 12:38 AM  -  TPS médio encontrado: 13.49090909090909

...

12/07/2008 12:39 AM  -  Tempo já decorrido nos trabalhos com checkpoint_segments: 3
minutos.
12/07/2008 12:39 AM  -  Tempo total permitido para trabalho neste item: 40 minutos.
12/07/2008 12:39 AM  -  Limite máximo de performance encontrado nas configurações
atuais para o item checkpoint_segments.
12/07/2008 12:39 AM  -  A partir deste ponto a performance não melhora para a carga
fornecida. No próximo dia, no entanto,
12/07/2008 12:39 AM  -  com o novo conjunto de configurações e carga este item pode
voltar a ter o seu valor alterado.
12/07/2008 12:39 AM  -  Fim dos testes diários com o item checkpoint_segments.
12/07/2008 12:39 AM  -  ------------------------------------------------------
12/07/2008 12:39 AM  -  Iniciando trabalhos com shared_buffers

...

12/07/2008 12:41 AM  -  Valor anterior encontrado: 24MB
12/07/2008 12:41 AM  -  Novo valor calculado: 32MB

...

12/07/2008 12:42 AM  -  TPS médio encontrado: 13.745454545454544
12/07/2008 12:42 AM  -  Tempo já decorrido nos trabalhos com shared_buffers: 2
minutos.
12/07/2008 12:42 AM  -  Tempo total permitido para trabalho neste item: 40 minutos.

...

12/07/2008 12:45 AM  -  Iniciando trabalhos com effective_cache_size

...

12/07/2008 12:46 AM  -  Valor anterior encontrado: 96MB
12/07/2008 12:46 AM  -  Novo valor calculado: 128MB

...

12/07/2008 12:48 AM  -  TPS médio encontrado: 14.0
```