PUC

ISSN 0103-9741

Monografias em Ciência da Computação
n° 41/08

Estado da Arte em Auto-Sintonia do Projeto Físico de Banco de Dados

José Maria Monteiro
Sérgio Lifschitz
Ângelo Brayner

Departamento de Informática

PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO DE JANEIRO
RUA MARQUÊS DE SÃO VICENTE, 225 - CEP 22453-900
RIO DE JANEIRO - BRASIL

Monografias em Ciência da Computação, No. 41/08 ISSN: 0103-9741
Editor: Prof. Carlos José Pereira de Lucena Julho, 2008

Estado da Arte em Auto-Sintonia do Projeto Físico de Banco de Dados

José Maria Monteiro, Sérgio Lifschitz, Ângelo Brayner[1]

[1] Mestrado em Informática Aplicada - Universidade de Fortaleza (UNIFOR)

monteiro@inf.puc-rio.br, sergio@inf.puc-rio.br, brayner@unifor.br

Abstract. Database physical design plays a critical role regarding performance. There has been considerable work on automated physical design tuning for database systems. Existing solutions require offline invocations of the tuning tool and strongly depend upon the capacity of DBAs to identify significant workloads manually. In real-world dynamic environments, with various ad-hoc queries, it is difficult to identify potentially useful indexes in advance. A few initiatives present brief descriptions of prototypes that address some aspects of online physical tuning. This work presents a partial survey in automated database physical design. We introduce a new taxonomy for the research efforts in autonomic databases. We give a detailed comparative analysis among the approaches for the automated database physical design available in the literature.

Keywords: Self-Tuning, Indexes, Database Tuning, DBMSs, Physical Design

Resumo. O desempenho dos servidores de bancos de dados é fator chave para o sucesso das aplicações de missão-crítica. Neste contexto, o projeto físico de bancos de dados desempenha um papel primordial para assegurar um desempenho adequado. Recentemente, algumas iniciativas apresentaram descrições de protótipos que implementam certas funcionalidades de sintonia automática do projeto físico. Neste artigo, apresentamos o estado da arte em auto-sintonia do projeto físico de banco de dados. Além disso, este trabalho propõe uma nova taxonomia para a classificação das pesquisas em auto-sintonia e realiza uma análise comparativa detalhada entre as principais propostas, encontradas na literatura, para a sintonia automática do projeto físico.

Palavras-chave: Auto-Sintonia, Índices, Ajustes de Desempenho, SGBDs, Projeto Físico

In charge for publications:
Rosane Teles Lins Castilho
Assessoria de Biblioteca, Documentação e Informação
PUC-Rio Departamento de Informática
Rua Marquês de São Vicente, 225 - Gávea
22453-900 Rio de Janeiro RJ Brasil
Tel. +55 21 3114-1516 Fax: +55 21 3114-1530
E-mail: bib-di@inf.puc-rio.br
Web site: http://bib-di.inf.puc-rio.br/techreports/

1 Introdução

As aplicações de bancos de dados têm se tornado cada vez mais complexas e variadas. Atualmente, estas aplicações podem ser caracterizadas pelo grande volume de dados e pela elevada demanda com respeito ao tempo de resposta das consultas e à vazão (*throughput*) das transações. Neste contexto, a sintonia fina do projeto físico de bancos de dados tem se revelado ainda mais importante, uma vez que esta desempenha um papel vital no desempenho dos sistemas de bancos de dados.

A sintonia do projeto físico de bancos de dados envolve a manutenção (criação, remoção e reorganização) de diferentes estruturas, tais como índices, visões materializadas e atividades como o particionamento de tabelas. Este fato decorre do grande benefício que estas estruturas podem trazer para o desempenho dos sistemas de bancos de dados, podendo reduzir substancialmente o tempo de execução das consultas, e até mesmo das atualizações [43].

Realizar o ajuste do projeto físico de forma manual tem se tornado uma tarefa extremamente complexa para as aplicações atuais. Esta tarefa requer um profundo conhecimento acerca dos detalhes de implementação dos Sistemas de Gerenciamento de Bancos de Dados (SGBDs), das características dos dados armazenados, das aplicações e da carga de trabalho (conjunto de consultas e atualizações) submetida ao SGBD [7, 29].

A manutenção do projeto físico de bancos de dados é uma das tarefas mais complexas, e das que consomem mais tempo, de um DBA (*Database Administrator*). Contudo, esta não é a única atividade desempenhada por este profissional. As atividades do DBA envolvem o planejamento da capacidade de *hardware*, a garantia da segurança lógica dos dados, o ajuste das configurações dos recursos gerenciados e o gerenciamento das dependências inter-sistemas (como por exemplo, entre o *middleware* e o SGBD) [32]. Logo, fornecer ferramentas que auxiliem o DBA na difícil e repetitiva tarefa de ajustar o projeto físico torna-se de fundamental importância.

Atualmente, os principais fabricantes de SGBDs (*Oracle*, *IBM* e *Microsoft*) oferecem ferramentas para suportar a sintonia do projeto físico de bancos de dados [19, 48, 4]. Estas ferramentas auxiliam os administradores (DBAs) através da análise automática da carga de trabalho e, com base nesta análise, recomendam ações como a criação/remoção de índices.

Tais ferramentas adotam uma abordagem *offline* (descontínua) na solução do problema do projeto físico e transferem para o DBA a decisão final sobre os ajustes a serem realizados. Especificamente, os DBAs necessitam capturar uma amostra representativa da carga de trabalho e fornecer esta amostra para a ferramenta de sintonia. Além disso, os DBAs precisam *adivinhar* o momento em que o sistema necessita de ajustes, a fim de iniciar uma sessão de sintonia fina (executando explicitamente a ferramenta). Em seguida, os DBAs ainda necessitam decidir se devem ou não executar as recomendações sugeridas pela ferramenta e, principalmente, qual o melhor momento para executar estas recomendações, a fim de não comprometer o desempenho do sistema, uma vez que a criação de índices e/ou visões materializadas podem consumir recursos de processamento, por exemplo.

Adicionalmente, a tarefa de modelar uma carga de trabalho em cenários dinâmicos é por demais complexa, e é igualmente difícil decidir quando o sistema de banco de dados necessita de ajustes. Utilizar a ferramenta de sintonia e executar os ajustes de forma freqüente resulta em desperdício de recursos, podendo degradar o desempenho do sistema. Por outro lado, executar as tarefas de sintonia esporadicamente pode conduzir à perda de

importantes oportunidades de melhora de desempenho [37].

Outra desvantagem das ferramentas de sintonia automática (*wizards* ou *advisors*) consiste na estratégia de analisar a carga de trabalho coletada durante um intervalo de tempo fixo e, com base nesta análise, criar uma configuração de índices estática, ou seja, que não será alterada até que uma nova sessão de sintonia fina seja executada pelo DBA. Todavia, na grande maioria das aplicações recentes a utilização do banco de dados muda com o tempo, isto é, as características da carga de trabalho submetida ao SGBD podem mudar (OLTP no início do mês e OLAP ao final do mês, por exemplo), o acesso aos dados pode variar de forma sazonal (apresentando picos de acesso em determinados períodos do dia ou do mês), novas aplicações que fazem acesso aos mesmos dados podem ser colocadas em produção, etc. Assim, a configuração adequadamente escolhida em um determinado momento pode deixar de ser boa em um instante futuro [37]. Conseqüentemente, em ambientes dinâmicos, com várias consultas *ad-hoc*, torna-se bastante complicado identificar os índices potencialmente úteis, mesmo com o auxílio das ferramentas de sintonia.

Podemos observar que a sintonia do projeto físico não é um processo estático, que se realiza uma única vez, mas, ao invés disso, consiste em um processo no qual os DBAs necessitam monitorar continuamente a carga de trabalho, e analisar esta carga, buscando diagnosticar problemas de desempenho e, caso necessário, realizar os ajustes adequados. Logo, para a maioria das aplicações de bancos de dados a sintonia do projeto físico deve ser realizada de forma interativa e contínua [29].

Recentemente, algumas iniciativas apresentaram descrições de protótipos que implementam algumas funcionalidades na direção da sintonia automática [37, 38, 16, 33, 41, 40, 29]. Todavia, estes trabalhos:

- adotam uma abordagem intrusiva (ou seja, exigem alterações no código fonte do SGBD),
- funcionam apenas com um SGBD específico,
- requerem chamadas adicionais ao otimizador para obter informações acerca dos índices candidatos,
- não fornecem solução para o problema da estabilidade da configuração escolhida,
- ignoram o problema da fragmentação das estruturas de índices e
- consideram somente a utilização de índices secundários.

Além disso, pesquisas recentes realizadas pelos principais fabricantes de SGBDs indicam que o gerenciamento autônomo do projeto físico de bancos de dados está ganhando grande aceitação como uma importante direção para o desenvolvimento de futuras ferramentas comerciais [7, 14].

Neste artigo, apresentamos o estado da arte em auto-sintonia do projeto físico de banco de dados. Além disso, este trabalho propõe uma nova taxonomia para a classificação das pesquisas em auto-sintonia e realiza uma análise comparativa detalhada entre as principais propostas, encontradas na literatura, para a sintonia automática do projeto físico.

Este trabalho está organizado da seguinte forma. A Seção 2 discute uma nova taxonomia para a classificação das pesquisas anteriormente realizadas na área de sintonia automática. Na Seção 3 são apresentadas as abordagens relacionadas à auto-sintonia

global. Já a Seção 4 descreve as principais propostas para solucionar um dos problemas mais relevantes na área de auto-sintonia local: a manutenção automática do projeto físico. A Seção 5 conclui este trabalho e aponta direções para trabalhos futuros.

2 Classificação das Pesquisas em Auto-Sintonia de Bancos de Dados

Um estudo detalhado das pesquisas em auto-sintonia de bancos de dados presentes na literatura e em sistemas comercias até 2004 é apresentado em [26]. Este trabalho descreve os princípios nos quais baseia-se a Auto-Sintonia e propõe uma classificação das linhas de pesquisa de acordo com o foco dado na abordagem do problema, ou seja, de acordo com a abrangência da solução. Desta forma, os trabalhos de pesquisa existentes foram classificados em dois grandes grupos: Auto-Sintonia Local e Auto-Sintonia Global.

No primeiro grupo (auto-sintonia local) a ênfase se dá em estudar problemas específicos de sintonia existentes nos SGBDs atuais. Dentre estes problemas podemos citar: o projeto físico de bancos de dados (em especial, a seleção de índices), a alocação de dados entre diferentes elementos de armazenamento, o controle de carga, o refino de estatísticas utilizadas pelo otimizador, a substituição de páginas e o ajustes de áreas de memória. Cada trabalho de auto-sintonia local enfoca um destes problemas, analisando vantagens e desvantagens de uma solução que reduz ou elimina a intervenção humana.

No grupo de auto-sintonia global procura-se estudar como é possível tomar ações de sintonia que tragam benefícios de desempenho para o sistema como um todo. Este tipo de abordagem procura encontrar um equilíbrio entre as diversas considerações locais de sintonia de forma a alcançar um desempenho global que seja melhor do que o que seria alcançado caso cada componente do sistema tomasse decisões de sintonia isoladamente. Ainda há poucos trabalhos nesta linha de pesquisa e, de fato, bem poucos trazem algum resultado experimental. Isto pode estar relacionado ao fato das inter-relações entre os diversos componentes de um SGBD não serem bem conhecidas [46].

Contudo, devido ao grande número de trabalhos publicados nos últimos anos e a grande variedade de ferramentas, estratégias e abordagens propostas, propomos aqui uma nova forma de classificar as linhas de pesquisa existentes na área de sintonia automática. Neste trabalho temos uma nova abordagem baseada em duas dimensões: (1) a abrangência da solução (Global vs Local) - a dimensão X - e (2) o grau de independência da solução em relação ao código do SGBD (Não-Intrusiva vs Intrusiva) - a dimensão Y, a qual definiremos no decorrer desta seção. Podemos dispor esses eixos em um plano cartesiano, como mostra a Figura 1.

Na classificação proposta adicionamos uma nova dimensão: o grau de independência da solução em relação ao código do SGBD. Esta dimensão baseia-se nos conceitos de acoplamento e coesão. Acoplamento é o grau em que os módulos são relacionados ou dependentes de outros módulos. Já coesão é o nível de integridade interna de um módulo. Módulos com alta coesão têm responsabilidades bem definidas e são difíceis de dividir em dois ou mais módulos. Em [47], Yourdon e Constantine argumentam que projetos de *software* que possuem módulos com baixo acoplamento entre si e alta coesão dão origem a códigos mais confiáveis e fáceis de manter.

De acordo com esta nova dimensão, as soluções de auto-sintonia podem ser classificadas como: Intrusivas e Não-Intrusivas. As soluções intrusivas são aquelas que exigem alterações

Figura 1: Classificação das Pesquisas em Auto-Sintonia

no código do SGBD, ou seja, que estão fortemente acopladas ao código do SGBD. Observe que diferentes abordagens podem apresentar níveis de acoplamento (ou intrusividade) distintos, requerendo uma maior ou menor quantidade de alterações no código do SGBD. Já as soluções não-intrusivas são aquelas que não requerem modificações no código do SGBD, ou seja, estão desacopladas deste código. Logo, as soluções não-intrusivas não são afetadas por alterações ou atualizações na implementação do SGBD, ou seja, são independentes em relação ao lançamento de novas versões ou "*releases*", o que facilita sua evolução. Por outro lado, o código das abordagens intrusivas apresentam um forte acoplamento (além de baixa coesão) com o código do SGBD, sendo portanto altamente dependente deste, o que dificulta sua evolução. Neste caso, o lançamento de uma nova versão ou "*release*" do SGBD implica, geralmente, na necessidade de se alterar o código da solução intrusiva.

3 Auto-Sintonia Global de SGBDs

Em [31, 30] os autores propõem duas arquiteturas, baseadas em agentes de *software*, onde todos os parâmetros de sintonia de um SGBD seriam ajustados automaticamente, por meio destes agentes. Estes trabalhos partem do princípio que tratar a sintonia de um SGBD focando apenas em componentes locais, sem levar em conta a interação entre eles, não constitui uma abordagem eficiente. Isto significa que as interações poderiam causar prejuízos a parâmetros previamente ajustados.

O processo de auto-sintonia global busca, em última análise, uma configuração ótima. O problema pode ser formulado como Otimização Combinatória Multi-Objetivo, onde várias soluções eficientes seriam geradas e a escolha final ficaria a cargo do próprio Sistema. Tendo-se estabelecido um conjunto inicial de métricas, o Sistema seria capaz de escolher uma solução conveniente [33].

Antes de entrar em detalhes sobre cada arquitetura proposta, vale ressaltar as etapas constituintes de um processo de auto-sintonia genérico:

1. **Observação:** captam-se imagens do estado do Sistema em intervalos previamente configurados. Realizado de forma minimamente intrusiva, alimentará a base histórica, a ser consultada posteriormente caso surjam conflitos.

2. **Análise:** comparam-se resultados advindos da etapa Observação com métricas previamente armazenadas. Caso existam distorções, emite-se um alerta indicando a necessidade de realização de ajustes.

3. **Planejamento:** criam-se estratégias para resolução de gargalos identificados na etapa Análise. Destaca-se a importância de aplicar mudanças ao Sistema em períodos de baixa atividade, para que seja possível medir o salto qualitativo da técnica empregada com um mínimo de interferências.

4. **Ação:** acontecem as alterações decididas na etapa Planejamento.

Em [30] foram propostas duas arquiteturas para auto-sintonia global:

1. **Centralizada:** há um agente, denominado Coordenador, que concentra o poder decisório do Sistema. Os demais agentes limitam-se apenas a coletar informações e emitir alertas quanto a problemas em potencial. Interessante perceber que, tomando o *framework* sugerido em [24], este agente teria apenas as camadas de Crença, Raciocínio e Colaboração.

2. **Distribuída:** nesta abordagem, o conhecimento e o poder decisório estão distribuídos por todos os agentes do Sistema. Ainda assim, há necessidade de um agente especial, o Analisador, que faria o papel de "juiz" em caso de conflitos.

Os autores escolheram a arquitetura centralizada para realização de testes e obtenção de resultados. A implementação foi conduzida de forma acoplada ao núcleo do SGBD, ou seja, não existe separação entre o código do SGBD e o código do Sistema de Agentes [33]. Assim, podemos classificar esta abordagem como global e intrusiva.

Mais recentemente, os SGBDs comerciais vêm dando uma maior atenção à auto-sintonia global, como no caso do Oracle 10g [17]. Na versão 10g do SGBD da Oracle foi incluído um componente especializado em diagnóstico automático. Este componente considera em suas análises a vazão (*throughput*) total do sistema, ao invés de focar somente em problemas de desempenho locais. Entretanto, a idéia de usar um "*database time*" como unidade de performance para comparar diferentes problemas de desempenho e administração pode não capturar muitas situações comuns que acontecem na prática. O componente de diagnóstico automático foi implementado junto ao código do Oracle 10g. Logo, essa abordagem também pode ser classificada como global e intrusiva.

4 Auto-Sintonia Local de SGBDs

Nos últimos anos, diversos fabricantes de sistemas de gerenciamento de banco de dados têm disponibilizado ferramentas para a seleção automática de índices, as quais auxiliam o trabalho dos administradores de bancos de dados. No contexto do projeto AutoAdmin [18] da *Microsoft*, iniciado em 1997, Chaudhuri e Narasayya, em [11, 12, 13], apresentam os algoritmos e problemas enfrentados no desenvolvimento da ferramenta *Index Tuning*

Wizard para o *Microsoft SQL Server 7.0* onde, a partir de uma análise de uma carga de trabalho (*workload*), obtida pelo DBA, a ferramenta sugere a criação de um conjunto de índices. Em [13], os autores utilizam uma técnica de otimização denominada "*hill climbing*" como estratégia de busca. Trabalhos adicionais sobre as heurísticas de seleção de índices, realizados pelos mesmo autores, podem ser encontrados em [10]. Dois anos depois, os trabalhos evoluíram para a seleção automática, não apenas de índices, mas também de visões materializadas [2]. Acompanhando o lançamento da versão 2005 do *SQL Server*, o processo foi estendido incluindo a sugestão de particionamento em grandes tabelas [1].

Em [10] formaliza-se a seleção de índices como um problema de otimização e demonstra-se que, tanto a escolha de uma configuração ótima de índices secundários (*non-clustered*) como primários (*clustered*), são problemas NP-difícil. Também sugere-se uma heurística capaz de atribuir benefícios individuais a cada índice participante de um comando. Destaca-se também a preocupação em levar em conta a interação entre índices, ou seja, o benefício obtido por um índice composto às vezes supera a soma dos benefícios dos índices sobre os atributos constituintes.

Uma abordagem que simplifica e uniformiza o conjunto de técnicas e heurísticas anteriormente propostas para automação do projeto físico de bancos de dados constitui a maior contribuição de [4]. Neste trabalho, os autores propõem que a etapa de seleção de índices candidatos (que inclui a enumeração e o "*merge*" de índices candidatos) seja realizada durante, e em conjunto, com o processo de otimização da consulta, e não mais sejam realizada separadamente e por heurísticas distintas. A principal vantagem dessa abordagem é a diminuição do número de estruturas candidatas. Além disso, os autores propõem uma estratégia diferente para a busca da configuração de índices e visões materializadas. Ao invés de utilizar um algoritmo guloso, baseado no problema da mochila, que inicia com uma configuração vazia e vai adicionando novas estruturas até que o limite de espaço disponível seja alcançado, a abordagem proposta começa com uma configuração ótima, mesmo que o espaço utilizado nesta configuração ultrapasse o espaço disponível. Em seguida, vai "relaxando" a solução inicial de forma progressiva gerando novas configurações que consomem menos espaço que as configurações anteriores. Para gerar estas novas configurações as heurísticas propostas podem remover ou juntar um sub-conjunto das estruturas selecionadas. Este processo é repetido até que seja gerada uma configuração que satisfaça a restrição de espaço.

Os algoritmos gulosos utilizados para a busca da melhor configuração de índices e visões materializadas nem sempre conseguem obter uma solução ótima. O trabalho apresentado em [35] propõe um modelo para a seleção de índices baseada na Programação Linear Inteira (*Integer Linear Programming - ILP*). A utilização da ILP possibilita a utilização de uma grande quantidade de técnicas de otimização combinatória a fim de proporcionar a obtenção de garantias de qualidade, de soluções aproximadas e até mesmo soluções ótimas. A abordagem apresentada proporciona soluções de alta qualidade, eficiência e escalabilidade.

Em [25], os autores propõem uma forma alternativa de análise de cargas de trabalho, baseada em amostragens, com o objetivo de diminuir o "*overhead*" gasto neste processo.

Já [5] apresenta o problema de refinamento do projeto físico. Este problema ocorre quando o DBA não consegue coletar ou produzir uma amostra significativa da carga de trabalho. Neste cenários, as ferramentas de sintonia automática não poderiam ser uti-

lizadas, uma vez que esta amostra é um pré-requisito para o funcionamento de tais ferramentas. Contudo, o DBA está apto a fornecer um projeto físico inicial, baseado em seu conhecimento. Porém, esta configuração inicial pode violar restrições importantes, como, por exemplo, o espaço disponível para armazenamento. A abordagem proposta em [5] parte de uma configuração inicial fornecida pelo DBA e progressivamente refina esta solução até chegar em uma nova solução que satisfaça as restrições existentes (como o espaço disponível, por exemplo). Este processo de refinamento baseia-se em dois novos operadores, denominados "*merging*" e "*reduction*", e trata a seleção de índices, visões materializadas e índices definidos sobre visões materializadas de forma unificada.

A IBM também possui linhas de pesquisas voltadas para a busca da automação completa do gerenciamento de bancos de dados. Destaque para o projeto SMART [20], que visa enriquecer o gerenciador DB2 com características autonômicas. Em [28], Lohman et al. descrevem um algoritmo baseado no clássico problema da mochila que foi utilizado na ferramenta de seleção de índices do IBM DB2. Este trabalho sugere que a heurística para seleção de índices deve ser intimamente integrada com o otimizador. Neste sentido, o otimizador é estendido com um modo de sugestão de índices e, antes da otimização de uma determinada cláusula SQL, são gerados índices hipotéticos, para todas as colunas relevantes. Em seguida, os índices recomendados para a cláusula SQL são utilizados como entrada em uma heurística de seleção de índices que tenta encontrar o melhor conjunto de índices para a carga de trabalho como um todo. A heurística para seleção de índices candidatos apresentada neste trabalho foi implementada em [37, 16, 38, 33, 27]. Finalmente, Lohman et al. [48] apresentam uma abordagem para a seleção automática de visões materializadas. As pesquisas desenvolvidas pela IBM e discutidas até aqui podem ser classificadas como abordagens locais e intrusivas.

A Oracle também tem investido bastante nos últimos anos buscando adicionar funcionalidades de auto-sintonia em seu SGBD. A versão do SGBD, Oracle 10g já oferece vários recursos que permitem a criação de *scripts* contendo instruções que dinamicamente ajustem o ambiente, segundo os níveis de utilização [9]. Já em [19], discute-se o funcionamento da ferramenta ADDM (*Automatic Database Diagnostic Monitor*). Esta ferramenta permite realizar o acompanhamento e o ajuste de desempenho de forma automática em bancos de dados Oracle [33]. Estas iniciativas também podem ser classificadas como locais e intrusivas.

O SGBD de código aberto PostgreSQL oferece poucos recursos de auto-sintonia, com destaque para a ferramenta pg_autovacum [36]. Esta ferramenta, periodicamente, compacta tabelas eliminando fisicamente *tuplas* que tenham sido marcadas para exclusão [33].

4.1 Outros casos relevantes

O modelo de sintonia fina utilizado nos trabalhos anteriores assume que uma carga de trabalho consiste de um conjunto de consultas e atualizações. Contudo, [3] mostra que se for possível representar uma carga de trabalho como uma seqüência, ou uma seqüência de conjuntos, explorando a ordem das cláusulas SQL na carga de trabalho, pode-se ampliar a utilização das ferramentas de sintonia, obtendo significativos ganhos de desempenho. O problema em questão foi formalmente definido e foi apresentada uma abordagem ótima para a sintonia de seqüências, a qual baseia-se no problema do caminho mínimo, que se mostrou bastante eficiente na prática. As técnicas propostas em [3] foram implementadas e avaliadas através de uma extensão do *Database Tuning Advisor*, o qual faz parte do

Microsoft SQL Server 2005.

Outro problema importante consiste na variação da carga de trabalho. Quando o padrão da carga de trabalho ou as características dos dados requisitados sofrem alterações o projeto físico anteriormente selecionado pelas ferramentas de sintonia pode não mais ser adequado. Neste caso, o DBA necessita capturar uma nova amostragem da carga de trabalho e executar a ferramenta novamente. Contudo, o DBA não sabe a princípio quando a carga de trabalho sofrerá alterações ou mesmo se a carga corrente está sofrente modificações em suas características. Por outro lado, monitorar constantemente o comportamento da carga de trabalho pode ser inviável em termos de custo de processamento. Para enfrentar este problema, o trabalho apresentado em [6] propõe que as ferramentas de sintonia possuam um módulo de alarme ("*alerter*") que notifiquem o DBA quando oportunidades (ou necessidades) significativas de sintonia ocorram. O protótipo descrito em [6] foi divido em dois componentes: um cliente, escrito em C++ e outro servidor, implementado junto ao código do SQL Server 2005.

Em [14], os autores discutem os avanços na área de auto-sintonia obtido no período de 1997 a 2007, baseando-se principalmente nas experiências do projeto *AutoAdmin* e focando principalmente no problema do projeto físico automático. Contudo, avanços em outras áreas da auto-sintonia foram brevemente discutidos. Além disso, destaca-se que em bancos de dados realmente grandes ("*Very Large Databases*") qualquer alteração no projeto físico é sempre uma operação bastante demorada ("pesada"). Recentemente, têm sido propostas algumas abordagens mais leves quanto à definição das estruturas utilizadas no projeto físico, como por exemplo índices e visões materializadas parciais e "database cracking" [23, 22]. Vale ressaltar que estas mudanças implicam na revisão das abordagens para o projeto físico. Todos estes trabalhos de pesquisa realizados junto ao projeto AutoAdmin e discutidos até aqui podem ser classificados como soluções locais e intrusivas.

Todos os trabalhos discutidos até agora representam os avanços mais recentes na tecnologia de seleção automática de índices, visões materializadas, índices sobre visões materializadas e particionamento de grandes tabelas. Entretanto, tais abordagens não conseguem oferecer uma solução completa para a manutenção automática do projeto físico de bancos de dados (o que envolve estruturas de índices, visões e o particionamento de tabelas). Um ciclo completo para uma solução automática deveria conter: a coleta da carga de trabalho (*workload)* submetida ao banco de dados, a seleção das estruturas apropriadas (índices, visões e particionamentos) e a manutenção (criação, destruição ou reorganização) destas estruturas. Tudo isso de forma completamente automática, sem a intervenção humana.

Neste contexto, as soluções existentes (e anteriormente discutidas) necessitam da intervenção dos administradores de bancos de dados (DBAs), os quais nem sempre possuem todo o conhecimento e as ferramentas necessárias para caracterizar, de forma eficiente, a carga de trabalho submetida ao sistema. Assim, a decisão final sobre a criação ou remoção de uma determinada estrutura requer intervenção humana.

Alguns trabalhos têm sido propostos na direção de buscar uma solução completa para a manutenção automática do projeto físico. No caso do PostgreSQL, através da geração de protótipos, adicionando funcionalidades de auto-sintonia [37, 16, 38, 33, 27, 41, 29]. A Microsoft também tem investido na tentativa de estender o SQL Server 2005 adicionando funcionalidades que possibilite a manutenção automática do projeto físico [7, 8].

Já em [39, 40] propõe-se um *middleware*, denominado *QUIET*, situado entre as aplicações e o SGBD DB2, que sugere, de forma automática, a criação de índices. Porém, esta

solução baseia-se em comandos proprietários do DB2, os quais não existem em outros SGBDs. Além disso, exige que todas as cláusulas SQL sejam enviadas para o *middleware* e não mais para o SGBD, o que implica na necessidade de reescrever todas as aplicações previamente existentes e na impossibilidade de gerenciar uma carga de trabalho que seja submetida diretamente ao SGBD.

Todavia, os trabalhos de pesquisa mais recentes, que procuram fornecer suporte para a manutenção automática do projeto físico de bancos de dados, também podem ser classificados como soluções locais e intrusivas. Vale destacar ainda que Shasha [43] demonstrou, através de testes práticos, que a não manutenção periódica de índices faz com que o desempenho seja degradado com o tempo . Porém, poucos trabalhos procuraram solucionar o problema da recriação automática de índices ("*reindex*").

A seguir, as abordagens encontradas na literatura relacionadas à manutenção automática do projeto físico, serão discutidas com maiores detalhes.

4.2 Criação e Remoção Automática de Índices

O Trabalho proposto em [37, 38, 16] apresenta uma *motor* que possibilita a criação e destruição automática de índices no PostgreSQL. A abordagem proposta baseia-se na integração entre agentes de *software* e os componentes do SGBD. O modelo de raciocínio adotado nos agentes utiliza heurísticas a fim de escolher e automaticamente criar ou remover índices durante o funcionamento normal do SGBD. Os autores implementaram as heurísticas propostas em uma camada de *software* acoplada ao PostgreSQL. Assim, podemos classificar esta abordagem como uma proposta de auto-sintonia local e intrusiva. Alguns dos problemas de implementação são discutidos nestes artigos, tais como a extensão do PostgreSQL para incluir o conceito de índices hipotéticos (virtuais) e o processo de sincronização necessário para integrar os agentes de *software* com o SGBD. Como principais contribuições deste trabalho, podemos citar: (i) uma extensão do PostgreSQL para incluir a noção de índices hipotéticos, e (ii) a implementação de um agente de auto-sintonia, integrado ao PostgreSQL, que possibilita a seleção e criação automática de índices. A remoção automática de índices foi proposta, mas não foi implementada e nem avaliada.

A idéia básica desta proposta consiste em utilizar duas heurísticas: uma para seleção de índices candidatos e outro para a seleção final de índices.

4.2.1 Heurística para Seleção de Índices Candidatos

A primeira heurística ocupa-se em analisar as cláusulas SQL submetidas ao SGBD, e enumerar índices que potencialmente melhorariam esta cláusula. Os índices selecionados por esta heurística (e que se supõem úteis) passam a existir como hipotéticos. Este processo de seleção de índices candidatos inspirou-se numa heurística proposta em [28]. Como esta presume a existência de índices do tipo "*what-if*", tais como propostos em [12], houve a necessidade de estender a funcionalidade do SGBD PostgreSQL para que o otimizador levasse em conta a presença de índices hipotéticos [38]. Comandos foram estendidos para que tais índices virtuais pudessem ser criados ou destruídos, e também para que aparecessem em planos de execução. Mostrou-se que a utilização deste tipo de índice não causou nenhuma sobrecarga ao SGBD, mesmo quando criados em tabelas volumosas.

Esta heurística, adaptada de [28], analisa predicados e cláusulas presentes no comando SQL submetido ao SGBD para encontrar colunas que poderiam ser indexadas. Colunas consideradas interessantes são classificadas e combinadas para formar índices hipotéticos - índices que existem apenas na metabase do SGBD, mas que não existem fisicamente.

A heurística procura encontrar no comando SQL cinco tipos interessantes de colunas:

1. **EQ:** colunas envolvidas em predicados de igualdade;
2. **O:** colunas envolvidas em clausulas ORDER BY, GROUP BY e predicados de junção;
3. **RANGE:** colunas que aparecem em restrições de intervalos;
4. **SARG:** colunas que aparecem em outros predicados indexáveis (por exemplo, like);
5. **REF:** demais colunas referenciadas no comando SQL.

Na implementação realizada em [37], foi identificado, ainda, mais um grupo interessante de colunas, denominado BAD. Este grupo é formado por colunas afetadas por comandos de atualização. Em comandos do tipo *insert* e *delete*, são todas as colunas das tabelas atualizadas; em comandos do tipo *update*, são as colunas referenciadas na cláusula *SET*. Este grupo de colunas seria interessante para determinar, posteriormente, quais índices são prejudicados pelo comando. A heurística, então, combina os grupos de colunas das seguintes formas para formar índices hipotéticos com múltiplas colunas, em ordem, eliminando colunas duplicadas:

- **EQ + O**
- **EQ + O + RANGE**
- **EQ + O + RANGE + SARG**
- **EQ + O + RANGE + REF**
- **O + EQ**
- **O + EQ + RANGE**
- **O + EQ + RANGE + SARG**
- **O + EQ + RANGE + REF**

Estas combinações procuram refletir usos típicos de índices para acelerar consultas [28]. Para identificar índices mais simples, por exemplo com uma coluna, a heurística original propõe, após a realização das combinações acima, que a lista de índices seja complementada com uma etapa de enumeração exaustiva de índices envolvendo as colunas selecionadas. Esta enumeração seria interrompida por um limite de tempo. Vale destacar que esta abordagem produz uma quantidade muito grande de índices candidatos. A maioria destes índices nunca serão materializados, porém terão que continuar sendo gerenciados, gerando sobrecarga (*"overhead"*).

Na implementação realizada em [37], onde um agente é executado de forma embutida no servidor, o uso deste tipo de estratégia exaustiva de enumeração pode representar um aumento considerável no consumo de recursos e um indesejável atraso na escolha de índices candidatos para o comando analisado. Assim, os autores limitaram a busca executada pela heurística à enumeração dos índices possíveis de uma coluna envolvendo os grupos de colunas EQ, O, RANGE e SARG. Estes grupos representam os usos de colunas em predicados que admitem indexação.

4.2.2 Heurística para Seleção Final de Índices (Heurística de Benefícios)

A segunda, denominada Heurística de Benefícios, procura decidir de forma "*online*", à medida que o sistema recebe novos comandos SQL, quais índices devem ser criados ou removidos para tornar o processamento da carga de trabalho como um todo mais eficiente.

Esta heurística acompanha os índices hipotéticos, atribuindo-lhes benefícios à medida que contribuam de forma positiva nos comandos executados pelo SGBD. Isto é conhecido uma vez que obtém-se o custo de cada comando e quanto este custo poderia cair, caso existissem os índices. Quando o benefício acumulado para um determinado índice for grande a ponto de compensar o custo da criação do índice, este deixa de ser hipotético e transforma-se automaticamente em índice real. Esta estratégia permite tomar decisões não somente de criação de índices candidatos, mas também de remoção de índices reais previamente existentes, que venham a deixar de ser úteis.

A heurística de benefícios foi inicialmente proposta em [15], porém nunca tinha sido testada para uso efetivo em um SGBD. Em [37] propõem-se diversas extensões ao trabalho apresentado em [15], tais como a definição de equações para cálculos dos benefícios, e refinamento das estratégias de avaliação de consultas, que tornaram a heurística mais realista.

A idéia básica é que todo índice participante de um comando receba o mesmo benefício de todos os índices presentes no comando. Esta abordagem pode criar distorções, já que um índice, que pouco tenha contribuído para baixar o custo de um comando, pode vir a ter um benefício acumulado falacioso [33].

Por exemplo, imagine que o custo de uma determinada consulta seja 2.000, mas, graças à utilização de dois índices, caia para 500. Suponha também que, dos 1.500 obtidos de benefício, 1.490 sejam devidos apenas ao primeiro índice. Ora, segundo [37], os dois índices receberiam o benefício de 1.500, o que criaria uma flagrante distorção, já que o índice, que contribuiu apenas com 10, ganharia um benefício desproporcional à sua contribuição [33].

Uma alternativa mais justa consistiria em atribuir a cada índice uma parcela de ganhos proporcional à sua contribuição. Ao invés de cada índice participante de um comando receber um benefício fixo, tal qual o ganho total do comando, como proposto em [37], seria concedido um valor variável que dependeria da real contribuição do tal índice ao comando no qual estaria participando [33].

O trabalho proposto em [10] apresenta um algoritmo que analisa uma carga de trabalho e propõe um conjunto de índices, cujo benefício seja máximo. O algoritmo proposto baseia-se no problema da mochila e atribui "pesos" a índices candidatos. Desta forma, associam-se a cada índice ganhos proporcionais a sua contribuição para reduzir o custo global da carga.

Estimativa de Custos

A heurística de benefícios baseia-se nos seguintes fatores:

1. C_R**:**

   o custo, gerado pelo otimizador, do melhor plano de execução sobre a configuração de índices reais.

2. C_H**:**

   o custo, gerado pelo otimizador, do melhor plano de execução sobre a configuração de índices hipotéticos e reais. Usualmente, os índices hipotéticos escolhidos para uso neste plano foram sugeridos pela heurística de escolha de candidatos, quando executada para este mesmo comando SQL.

3. C_N**:**

   o custo, gerado pelo otimizador, do melhor plano sobre uma configuração física que não contém qualquer índice (nem real, nem hipotético).

4. B_K**:**

   o benefício do índice K para o comando SQL sendo atualmente analisado. O benefício será calculado pela heurística de forma distinta para índices reais e hipotéticos.

5. BA_{CK}**:**

   o benefício acumulado do índice K para todos os comandos já processados. Novamente, a heurística atualizará o benefício acumulado de acordo com o fato de o índice ser real ou hipotético.

6. CC_K**:**

   o custo estimado de criação do índice K. Normalmente, este custo não é calculado pelo otimizador do SGBD. Assim, foi também necessário estimá-lo em nossa implementação.

7. C_A**:**

   o custo estimado de atualização de um índice durante o processamento de um comando de atualização. Normalmente, este custo não é calculado pelo otimizador do sistema e deve ser estimado em sintonia com o modelo de custos empregado no SGBD. Os autores definiram uma aproximação para o custo de atualização de um índice:

   $C_A = 2 \times \lceil \frac{r}{R} \rceil \times P + c \times r$

   Na fórmula, r é o número de registros atualizados pelo comando, R é o número de registros da tabela, P é o número de páginas da tabela e c é um coeficiente que relaciona, percentualmente, o custo de uma operação de E/S com o custo de CPU para processar um registro que esteja em memória. No PostgreSQL, cada E/S tem custo estimado igual a um e o coeficiente c tem um valor padrão de 1%.

   O primeiro termo da fórmula calcula o custo de E/S da atualização. Estima-se que serão atualizadas uma quantidade de páginas de índice proporcional à fração

de registros da tabela que foram atualizados multiplicada pelo tamanho da tabela em páginas. Lembramos que, para índices hipotéticos, estima-se que o tamanho em páginas do índice é idêntico ao tamanho em páginas da tabela. Multiplica-se o resultado por dois pois, no pior caso, precisa-se ler e escrever cada página atualizada. Além do custo de E/S, ainda somamos o custo estimado de CPU de processar o número de registros que foram atualizados pelo comando.

8. CC_I**:**

   custo de criação de um índice I. Os autores definiram uma aproximação para o custo de criação de um índice I:

   $CC_I = 2 \times P + c \times R \log R$

   Supõe-se uma política de criação de índices em que todas as páginas da tabela são lidas, ordenadas e então o índice é criado de uma forma *bottom-up*. O primeiro termo da fórmula leva em conta o custo de E/S de ler todas as páginas da tabela e de escrever todas as páginas do índice (para índices hipotéticos, a mesma quantidade estimada). Já o segundo termo estima o custo de ordenar todas as linhas da tabela em memória.

Heurísticas em Ação

Quando uma nova operação (cláusula SQL) é submetida ao SGBD, o agente é notificado sobre o novo comando submetido e aplica a heurística de seleção de índices candidatos (hipotéticos). Neste momento, o otimizador gera o melhor plano de acesso dada a configuração real de índices existente. Em seguida, se inicia a heurística de seleção final de índices, ou heurística de benefícios. O agente, interagindo com o otimizador, envia a consulta para ser otimizada uma segunda vez e obtém o melhor plano de execução dada uma configuração de índices hipotéticos e reais. Os índices hipotéticos utilizados neste plano de execução são considerados índices candidatos para o comando SQL submetido. Depois disso, o agente, novamente a partir de interação com o otimizador, envia a consulta para ser otimizada uma terceira vez e obtém o custo de processamento do melhor plano de execução, para o comando SQL, sobre uma configuração em que não há índices definidos. Após estes procedimentos, são obtidos os fatores C_R, C_H e C_N calculados para o comando SQL recebido.

Se o comando submetido for uma consulta, aplica-se o procedimento mostrado na Figura 2. Para cada índice candidato a criação, calcula-se o seu benefício e atualiza-se o seu benefício acumulado. O benefício é a diferença entre o custo da consulta com a configuração de índices reais e o seu custo usando uma configuração de índices reais e hipotéticos.

Após o cálculo do benefício, o agente irá materializar o índice candidato somente se o seu benefício acumulado tiver superado o seu custo estimado de criação. Esta estratégia tem por objetivo criar índices cujo uso repetido se prove interessante o suficiente para compensar seu custo de criação.

Reparar que uma suposição implícita da heurística é a de que índices necessários para consultas passadas continuarão a ser interessantes para as consultas a serem processadas pelo SGBD no futuro. Assim, o agente tende a encontrar bons projetos de índices para cargas de trabalho relativamente estáveis. Vale ressaltar também que não existe preocupação

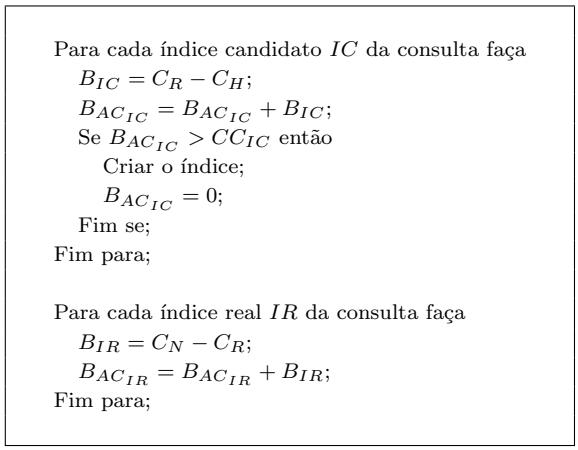

Figura 2: Estratégia de Avaliação de Consultas.

em relação ao espaço físico disponível, ou seja, a abordagem não considera a restrição de espaço. Contudo, na prática, esta restrição é extremamente importante uma vez que não existe espaço físico de tamanho infinito, mas muito pelo contrário, os dispositivos de armazenamento possuem capacidade limitada.

Para cada índice real, a heurística calcula o benefício como sendo a diferença entre o custo de processar o comando em uma configuração em que não há índices definidos e o custo na configuração de índices reais. Como o comando submetido é uma consulta, apenas atualiza-se o benefício acumulado para o índice real.

Se o comando submetido ao SGBD for uma atualização, aplicaremos o procedimento mostrado na Figura 3. Este procedimento se inicia com uma avaliação idêntica à realizada para consultas. É interessante observar que comandos de atualização devem inicialmente trazer os dados para a memória, exatamente como consultas, para somente então processar modificações sobre os dados. Normalmente, os otimizadores de SGBDs relacionais oferecem estimativas de custos para recuperar os dados necessários à atualização, mas não para os custos envolvidos na modificação propriamente dita dos dados.

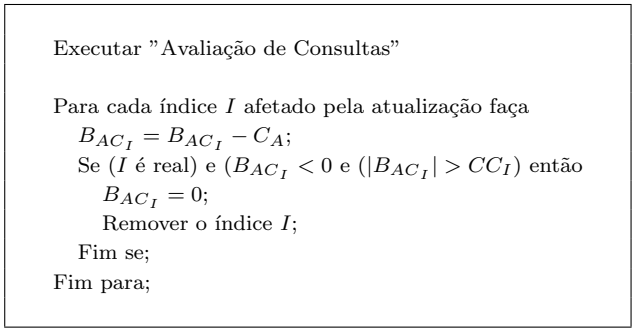

Figura 3: Estratégia de Avaliação de Atualizações.

Após a aplicação do mesmo procedimento para avaliação de consultas, contabiliza-se os custos de manutenção dos índices. Estes custos de manutenção serão debitados do

benefício acumulado tanto de índices candidatos quanto de índices reais. Para estimativa do custo de manutenção de índices, os autores introduziram o fator C_A. Em princípio, supõe-se que o custo de atualizar qualquer índice afetado pelo comando é idêntico e igual a C_A.

Após o cálculo do novo valor para o benefício acumulado de um índice real, verifica-se se a sua manutenção na base permanece vantajosa. Caso o benefício trazido pelo índice seja negativo e superior, em módulo, ao custo estimado de criação do índice, o mesmo é removido. Novamente, supõe-se que índices prejudiciais para comandos processados no passado continuarão sendo prejudiciais ao processamento de comandos pelo SGBD no futuro.

4.2.3 Implementação do Protótipo

Neste trabalho foram implementadas extensões para simulação de índices hipotéticos no SGBD objeto-relacional PostgreSQL versão 7.3[1]. Para isto, foi necessário implementar um mecanismo para registro dos índices hipotéticos no catálogo e alterar o otimizador de consultas para reconhecer as configurações hipotéticas registradas.

A linguagem de definição de dados do SGBD foi estendida para conter os seguintes novos comandos:

1. create hypothetical index;
2. drop hypothetical index ;
3. explain hypothetical ;
4. explain noindices.

Os primeiros dois comandos têm como finalidade possibilitar o registro e a remoção de índices hipotéticos. Estas definições são armazenadas no próprio catálogo, que foi estendido para permitir diferenciar índices hipotéticos de índices reais. O terceiro comando listado acima é uma extensão do comando *explain* do PostgreSQL. Ele permite que sejam obtidos planos de execução e custos de consultas levando em consideração os índices hipotéticos definidos. Já o quarto comando apresentado consiste em uma segunda extensão do comando *explain* do PostgreSQL. Esse comando permite que seja gerado um plano de execução ignorando completamente a existência dos índices (reais ou hipotéticos).

Vale ressaltar que foram realizadas alterações no catálogo do sistema, no otimizador de consultas, no processador de consultas, além da criação de quatro novos comandos. Todas essas alterações e extensões foram realizadas junto ao código do PostgreSQL versão 7.3, de forma intrusiva. Logo, ao surgir uma nova versão do SGBD PostgreSQL será necessário refazer todas estas alterações junto ao código fonte desta nova versão. Contudo, refazer todo esse esforço a cada lançamento de uma nova versão do SGBD pode se tornar impraticável.

[1]Em todo o ano de 2008 foram feitas adaptações para executar nas versões 8, que sofreu modificações significativas

4.2.4 Resultados Obtidos

A construção da carga de trabalho ocorreu graças ao *toolkit Database Test 2* (DBT-2) provido pela organização *Open Source Development Labs* [34]. Este *toolkit* simula uma carga de trabalho baseado no *benchmark* TPC-C [44], que favorece a incidência de curtas transações que consultam ou alteram pequenas quantidades de *tuplas*, simulando um ambiente com características OLTP (*On Line Transaction Processing*) [37].

Os resultados obtidos revelaram a eficácia da implementação. Mesmo capturando apenas cerca de 5% dos comandos executados no SGBD, o Agente de Benefícios foi capaz de analisar os mais representativos, já que todos os índices importantes foram criados. Além disso, este agente construiu um índice que não faz parte do conjunto de índices propostos pelo *benchmark* TPC-C. Os resultados apresentados em [37] mostram que este índice efetivamente melhorou o desempenho de algumas consultas, sendo, na prática, bastante útil.

Uma preocupação recorrente ao desenvolver componentes de *software* que atuem de forma concorrente aos serviços oferecidos pelo SGBD, consiste em garantir que não sejam intrusivos a ponto de comprometer o desempenho do Sistema. Pelos relatos apresentados em [37] não se observou nenhuma queda significativa no desempenho do SGBD que tenha sido causada pela presença do Agente de Benefícios.

4.2.5 Comentários Gerais

Esta abordagem pode ser classificada como local e intrusiva. A implementação gerada a partir das discussões presentes em [37] para o Agente de Benefícios, não contemplou a eliminação de índices reais, restringiu o uso de índices hipotéticos a árvores B^+, limitou-se a índices secundários e não considerou as restrições quanto ao espaço disponível para o armazenamento das estruturas de índices.

Além disto, deve-se frisar que a quantidade de índices hipotéticos criados é muito maior que a de índices reais, graças à Heurística de Enumeração de Índices Candidatos, o que gera uma sobrecarga ("*overhead*") desnecessária no gerenciamento das estruturas hipotéticas. Adicionalmente, quatro novos comandos tiveram de ser criados e diversas outras alterações tiveram que ser realizadas junto ao código do PostgreSQL versão 7.3. Todas essas alterações precisão ser refeitas a fim de portar o agente de benefícios para uma nova versão do PostgreSQL.

4.3 Recriação e Remoção Automática de Índices

Em [33], os autores estendem os trabalhos propostos em [37, 38, 16] em várias maneiras. Primeiramente, utilizando uma carga de trabalho diferente, baseada no TPC-H. Segundo, fazendo o acompanhamento dos índices previamente criados. Finalmente, implementando e avaliando a remoção automática de índices e, principalmente, recriando índices usando um *fill-factor* diferente nos nós folha. Diferentemente das ferramentas comerciais, o protótipo implementado não somente sugere um conjunto de índices a serem criados, mas também elimina e recria índices usando heurísticas próprias. Para avaliar as idéias propostas foi utilizada uma carga de trabalho baseada no TPC-H, mas com algumas alterações a fim de aumentar o número de atualizações (*updates*) e forçar a recriação dos índices.

Já foi aqui discutido que em [37] propõe-se que todo índice participante de um comando, receba o mesmo benefício de todos os índices presentes no comando. Uma alternativa mais justa consistiria em atribuir a cada índice uma parcela de ganhos proporcional à sua contribuição. O trabalho em [10] explica uma heurística capaz de calcular precisamente quanto cada índice contribuiu para redução de custos de uma consulta.

Já [33] adota uma estratégia intermediária, a qual baseia-se no acompanhamento dos índices criados e na atribuição de um bônus. Os autores decidiram realizar o acompanhamento de índices criados da seguinte forma: durante a fase como hipotético, enquanto o índice vai acumulando benefícios, também registra-se a quantidade de vezes em que ele foi útil. No momento em que deixa de ser hipotético para ser real, o índice ganha um "bônus" resultante da divisão entre o benefício acumulado, ou seja, igual ou um pouco superior ao custo de criação, pela quantidade de vezes em que foi utilizado.

Por exemplo, para o comando a seguir:

```
select prodnum, data, sum(valor) as total
from venda
where valor > 2000000
      and data between '20040101' and '20040102'
group by prodnum, data;
```

Apurou-se um custo de 60.000, mas a criação de um índice sobre o atributo data proporcionou uma economia de 10.112. Após a sexta execução do comando acima, o índice candidato acumulou um benefício de 60.672, portanto, justificando sua criação. E, como foi utilizado seis vezes, ganhou um bônus de 10.112. Durante sua fase como índice criado, toda vez que seja utilizado, seu benefício acumulado receberá um valor idêntico ao bônus. Desta forma, após a terceira utilização, seu benefício já estaria em 91.008 (60.672 + 3 x 10.112). Vale ressaltar que os benefícios apurados para um índice candidato podem variar de um comando a outro, já para os índices reais o benefício a ser acrescido tem sempre o valor do bônus [33].

Ainda que mencionada em [37], a eliminação automática de índices não foi implementada e nem avaliada. Em [33] esta funcionalidade foi implementada, porém, realizando uma pequena alteração. Segundo a Heurística de Benefícios [37], a decisão de eliminar um índice seria tomada quando os benefícios acumulados alcançassem valores negativos que, em módulo, superassem os custos para criação mais o ônus da destruição do índice. Este ônus, entretanto, revelou-se insignificante nos testes práticos, já que pôde-se concluir que trata-se, em última instância, da exclusão de uma *tupla* na metabase. Desta forma, desconsiderou-se este ônus no cálculo do benefício que indica a necessidade de destruir um índice.

Além disso, em [33], os autores decidiram acrescentar um detalhe de caráter prático: uma vez excluído um índice real, ele volta a ser hipotético, porém, seu benefício acumulado começa com uma carga negativa equivalente ao módulo de seu custo de criação. Esta decisão corrige a distorção de equiparar índices hipotéticos, que nunca causaram malefícios, com outros que já tenham prejudicado comandos durante a fase como índice real.

A partir dos experimentos realizados em [33] observou-se que o aumento da duração dos testes levava à maior incidência de eliminações e criações. As numerosas recriações dos índices levantaram dois questionamentos:

```
Para cada índice hipotético da consulta faça:
   Atualize benefícios acumulados
   Incremente vezes utilizadas
   Caso benefícios acumulados superem custo de criação:
      Calcule bônus
      Crie índice real
      Elimine índice hipotético
Para cada índice real da consulta faça:
   Incremente benefícios acumulados com bônus
```

Figura 4: Estratégia de Avaliação de Consultas.

```
Para cada índice afetado pela atualização faça:
   Decremente benefícios acumulados
   Caso índice seja real:
      Caso índice deva ser removido:
         Remova o índice criado
         Crie índice hipotético
```

Figura 5: Estratégia de Avaliação de Atualizações.

- Já que um índice será recriado em um momento do futuro, terá sido a destruição uma boa escolha?
- É fato notório os malefícios causados pela fragmentação de um índice [33]. Não seria mais adequado reconstruir um índice, ao invés de removê-lo, evitando períodos com status de hipotético após a primeira criação?

Certamente a destruição não constitui boa escolha para índices úteis no futuro. Durante o período em que o índice não existe, as consultas que poderiam se beneficiar deles sofrerão com aumentos de custos de execução.

Neste sentido, [33] apresenta uma heurística para reconstrução automática de índices, a qual analisa casos de eliminação iminente e, caso julgue interessante, dispara a reconstrução do índice. Assim, esta heurística estabelece regras que permitem decidir se um índice deve ser reconstruído ou eliminado. Vale ressaltar que são considerados por ora apenas índices possuindo organização em árvore B^+ (*Btree*).

Antes de analisarmos esta heurística, necessitamos entender os seguintes fatores:

1. T_t**:**

   tamanho em *tuplas* de uma determinada tabela.

2. T_i**:**

   tamanho em blocos de um determinado índice.

3. **Razão de Fragmentação (R):**

   R = número de *tuplas* de uma tabela / quantidade de blocos de índice da tabela

ou seja,

R = T_t / T_i

4. R_i**:**

   razão de fragmentação obtida logo após a criação da tabela, quando não existe fragmentação.

5. R_a**:**

   razão de fragmentação obtida num instante de tempo T_a.

6. **Grau de Fragmentação:** A medida do grau de fragmentação de um índice poderia ser obtida comparando-se a razão de fragmentação em dois momentos: logo após a criação R_i e no momento em que deseja-se verificar se vale a pena aplicar uma reconstrução, ou simplesmente seu descarte R_a.

   O trabalho apresentado em [33] propõe a seguinte fórmula para cálculo do grau de fragmentação de um índice:

   $GrF = 100 - [(R_a/R_i) \times 100]$

7. **V:**

   número de varreduras nas quais o índice participou (foi utilizado) desde sua criação.

Uma vez constatada a fragmentação de um índice, causada por sucessivas ocorrências de *page splits* (divisões de páginas), caso decida-se recriá-lo, recomenda-se fazê-lo deixando uma margem para futuras atualizações. Normalmente SGBDs possuem mecanismos que permitem dosar quantos *bytes* podem ser gravados por bloco. No PostgreSQL o fator de preenchimento é definido no código fonte (*hard coded*) e está definido em 90% para páginas nível folha e 70% para as demais.

A Heurística de Reconstrução Automática de Índices apresentada em [33] determina um fator de preenchimento de páginas com base no histórico de operações de varreduras nas quais participou de forma positiva o índice em vias de reconstrução. Como este trabalho utilizou o PostgreSQL para realizar as implementações, houve a necessidade de estender os comandos create index, reindex index e reindex table para que aceitassem uma nova cláusula, denominada fillfactor, o qual pode receber valores entre 1 e 9. O menor valor significa que apenas um décimo de cada página será ocupada, enquanto que o maior sinaliza 90% de ocupação. Além disso foi necessário implementar uma nova função, denominada pgstatindex, que checa todas as páginas de um dado índice. Esta nova função serviu de base para o desenvolvimento de um outro comando para o PostgreSQL, denominado getsize. Este comando, dado o nome de um índice, checa a quantidade de *tuplas* da tabela correspondente e conta as páginas do índice. Estes dois dados permitem calcular as duas razões R_i (logo após sua criação) e R_a (uma vez detectada a possibilidade de eliminá-lo).

Este fator de preenchimento (*fillfactor*) é calculado segundo a fórmula:

$$F = 10 - [(V + 10)div10]$$

Quanto mais varreduras sobre um índice fragmentado, menor será o seu *fillfactor*, objetivando reduzir a incidência de *page splits*. Entretanto, deve-se ressaltar que, ao reduzir o *fillfactor*, diminui-se a quantidade de informações por bloco, levando à necessidade de maior alocação de páginas.

Quando a Heurística de Benefícios decide remover um determinado índice real, a Heurística de Reconstrução Automática de Índices é executada. Este heurística avalia os valores dos fatores: G_rF, V e T_i. Caso estes fatores atendam a determinados limites (*thresholds*), efetua-se a reconstrução. Porém, como os autores não conseguiram determinar valores ótimos para os fatores G_rF, V e T_i, preferiu-se deixá-los como parâmetros de configuração para serem ajustados livremente pelo DBA.

A obtenção de resultados para o SGBD PostgreSQL somente pôde ser concluída após realizar algumas alterações importantes. Primeiramente, houve a necessidade de se criar um novo comando denominado *evaluate*. Este comando, repassa uma consulta recebida como parâmetro de entrada ao Agente de Benefícios, contudo não realiza a fase de execução. Desta forma, torna-se possível avaliar todas as consultas, sem as etapas mais onerosas. Por exemplo, suponha o comando SQL ilustrado na Figura 6, que referencia a maior tabela do Esquema TPC-H:

```
evaluate select linenumber, quantity, shipdate
   from lineitem
   where orderkey = 200
   and linenumber = 2
```

Figura 6: Exemplo de execução do comando *evaluate*, cujo argumento (select), tem sua execução inibida.

4.4 QUIET: Criação Automática de Índices Guiada por Consultas

Em [39, 40] propõe-se uma ferramenta, *QUIET* (*Query-Driven Index Tuning*) que sugere a criação de índices a partir de um modelo de custos e estratégias de seleção próprias.

A ferramenta proposta consiste em um *middleware*, situado entre as aplicações (que enviam as consultas dos usuários) e o SGBD DB2, funcionando assim como um *proxy*. A Figura 7 ilustra a arquitetura do *QUIET*.

Nesta abordagem, toda consulta deve ser enviada ao *middleware* (*QUIET*) e não mais ao SGBD. Desta forma, a ferramenta captura a carga de trabalho. Cada consulta capturada é analisada, com o objetivo de selecionar e criar os índices apropriados antes da execução da consulta. Somente após esta análise a consulta é repassada ao SGBD, que pode agora utilizar os índices recém criados no processamento da consulta recebida. Assim, a decisão sobre a criação das estruturas de índice é executada automaticamente, em tempo de execução, sem intervenção do DBA, repetindo-se continuamente a cada consulta capturada e analisada. Por este motivo a estratégia é denominada "Sintonia de Índices Guiada por Consultas". A seguir descrevemos como uma consulta capturada (Q) é processada:

1. Para cada consulta Q capturada pelo *QUIET* determina-se um conjunto de índices

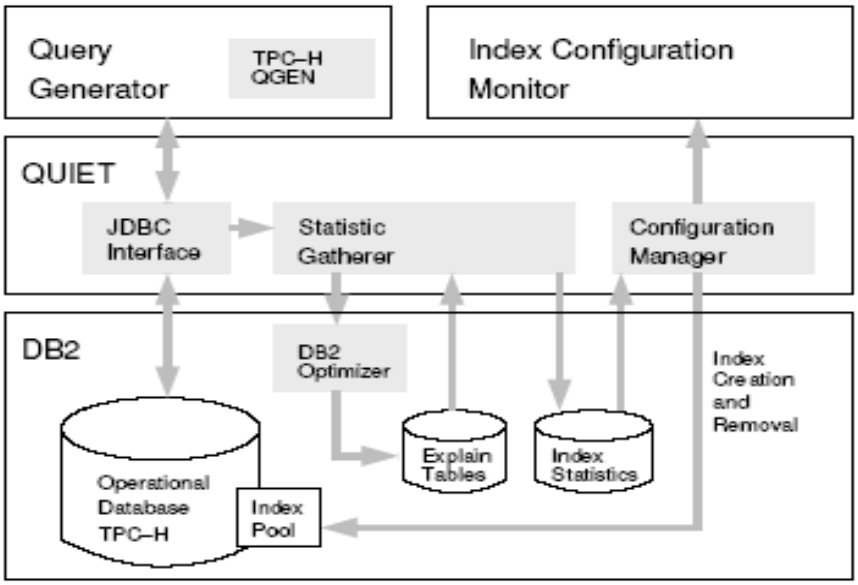

Figura 7: Arquitetura da Ferramenta QUIET

potencialmente úteis. Para isso utiliza-se uma estratégia extremamente semelhante à abordagem proposta em [28]. Estes índices são armazenados no catálogo do SGBD *DB2*, mais precisamente na tabela de sistema *ADVISE_INDEX*.

2. Em seguida, inicia-se a fase de análise da consulta Q. Esta análise baseia-se nas estatísticas acerca dos benefícios dos índices reais e hipotéticos (virtuais). Primeiramente a consulta Q é otimizada de forma convencional através da utilização do comando *EXPLAIN*. Como resultado deste comando é obtido o melhor plano de execução para a consulta Q, considerando-se somente os índices materializados (reais), e seu respectivo custo.

3. A consulta Q é otimizada uma segunda vez, agora considerando a existência tanto dos índices materializados quanto dos índices virtuais. Isso é conseguido através da utilização do comando EXPLAIN no modo RECOMMEND INDEXES (SET CURRENT EXPLAIN MODE RECOMMEND INDEXES), o qual é específico do *IBM DB2*. Neste modo, as consultas são compiladas, otimizadas e conjunto de índices é recomendado pelo otimizador. Os índices recomendados são denominados de virtuais (ou hipotéticos), uma vez que não existem fisicamente, sendo apenas uma sugestão do otimizador. Os índices sugeridos ficam armazenados na tabela de sistema (*ADVISE_INDEX*). Logicamente, esta abordagem não pode ser aplicada em SGBDs que não forneçam suporte a índices hipotéticos, como o *PostgreSQL* ou *SQL Server*.

4. Em seguida, calcula-se a diferença entre os custos dos planos gerados nas etapas (2) e (3). Este valor representa o benefício do conjunto de índices virtuais utilizados no plano de execução gerado em (3) ($profit(Q, I)$). Observe que a estimativa do benefício dos índices virtuais baseia-se no modelo de custos do próprio SGBD, logo esta é tão precisa quanto as estimativas do SGBD utilizado. Após atualizar o

benefício dos índices virtuais, obtém-se o conjunto dos índices com maior benefício e verifica-se a necessidade de alterar a configuração de índices corrente.

O valor obtido nesta etapa representa o benefício do conjunto de índices virtuais I. Porém é necessário estimar o benefício particular de cada índice $i \in I$, uma vez que os índices podem ter contribuído em proporções diferentes no custo do plano de execução gerado no passo (3). Para isso os autores sugerem as seguintes alternativas:

- Adicionar um valor constante ao benefício acumulado de cada índice $i \in I$.
- Adicionar o benefício total ($profit(Q, I)$) ao benefício acumulado de cada índice $i \in I$.
- Adicionar a média do benefício ($\frac{profit(Q,I)}{|I|}$) ao benefício acumulado de cada índice $i \in I$.
- Adicionar um benefício ponderado $profit(Q, I) \times \frac{profit(Q,\{i\})}{\sum_{j \in I} profit(Q,\{j\})}$ ao benefício acumulado de cada índice $i \in I$. Contudo, os autores não esclarecem como calcular $profit(Q, \{i\})$.

Utilizando esta abordagem, a ferramenta *QUIET* possibilita a criação automática de índices. Contudo, a criação indiscriminada de estruturas de índice pode consumir todo o espaço de armazenamento disponível. Para solucionar este problema, os autores utilizaram um "*Pool*" de índices, ou seja, um espaço de tamanho limitado para o armazenamento das estruturas de índices. O tamanho do "*Pool*" é determinado pelo DBA, através de um parâmetro do *middleware*.

4.4.1 Modelo de Custos

O conjunto de índices $I_1, \cdots, I_n$ utilizado durante o processamento de uma consulta Q é denominado I. O conjunto dos índices virtuais (hipotéticos) pertencentes a I é representado por $virt(I)$, já $mat(I)$ representa o conjunto dos índices materializados (reais) que pertencem a I. Seja $cost(Q)$ o custo de executar a consulta Q utilizando-se apenas os índices existentes (materializados), e $cost(Q, I)$ o custo de de se processar a consulta Q utilizando-se índices reais e hipotéticos (ou seja, I). Assim, o benefício de I para o processamento da consulta Q é definido como:

$$profit(Q, I) = cost(Q) - cost(Q, I)$$

Seja D o conjunto de todos os índices, reais e hipotéticos, existentes no catálogo do SGBD. O subconjunto de D contendo todos os índices reais em D é denominado configuração de índices (C). Logo, $C = mat(D)$. O tamanho de uma configuração é dado por:

$$\sum_{I_j \in I} size(I_j)$$

Vale ressaltar que o tamanho de uma configuração deve ser menor ou igual ao tamanho máximo do *pool* de índices (ou seja, do espaço reservado para as estruturas de índices). Logo:

$$\sum_{I_j \in I} size(I_j) \leq MAX\_SIZE$$

Com a finalidade de evitar freqüentes mudanças de configuração, uma alteração de configuração somente é realizada se a diferença entre o benefício da nova configuração (C_{new}) e o benefício da configuração atual (C_{curr}) ultrapassar um valor limite, denominado MIN_DIFF, o qual consiste em um parâmetro especificado pelo DBA. Assim:

$$profit(C_{new}) - profit(C_{curr}) > MIN\_DIFF$$

Em [40] os autores lançaram a idéia, e os primeiros resultados, da construção *on-the-fly* de índices. A idéia básica consiste em aproveitar a execução de um determinado operador, definido sobre uma determinada tabela, como por exemplo um TABLE SCAN, durante o processamento de uma determinada consulta, para construir um mais índices definidos sobre a mesma tabela. Os autores propuseram a criação de um novo comando denominado "CREATE DEFERRED INDEX", cuja sintaxe é descrita no exemplo a seguir.

```
CREATE DEFERRED INDEX idx_name
ON tabela (colunas)
```

Este comando registraria o novo índice (idx_name) no catálogo do SGBD, como um índice convencional, mas utilizaria um *flag* para indicar que este é um índice adiado (*deferred*), ou seja, um índice que ainda não foi fisicamente criado. Assim, adia-se a materialização do índice idx_name até que uma consulta SQL s qualquer, que envolva uma operação de TABLE SCAN sobre a mesma tabela na qual idx_name foi definido, seja executada. Assim, aproveita-se a execução da operação TABLE SCAN para materializar o índice idx_name, obtendo ganhos de desempenho, uma vez que não foi necessário realizar uma varredura na tabela apenas materializar o índice. Logo, para que esta abordagem fosse efetiva seria necessário implementar o comando "CREATE DEFERRED INDEX", uma vez que os SGBDs não possuem este comando, e reimplementar o operador TABLE SCAN. Porém, esta característica não pôde ser implementada no *QUIET* uma vez que esta ferramenta é construída sobre um SGBD comercial (DB2), cujo código fonte não é público. Foram realizados apenas algumas simulações utilizando o comando CREATE TABLE. Os autores também iniciam uma discussão sobre como implementar o comando "CREATE DEFERRED INDEX" e como estender o operador TABLE SCAN, a fim de permitir a materialização de uma ou mais estruturas de índices durante sua execução. Um protótipo com estas extensões foi implementado de forma intrusiva no SGBD PostgreSQL. Além disso, somente índices baseados em árvores B^+ foram considerados.

4.4.2 Comentários Gerais

Como esta abordagem não pressupõe alterações no código do SGBD ela poderia ser classificada como um estratégia local e não-intrusiva. Contudo, as consultas enviadas (pelos usuários ou aplicações) diretamente ao SGBD não serão capturadas pela ferramenta *QUIET*. Sendo, portanto, necessário alterar as aplicações legadas a fim de assegurar que todas as consultas sejam enviadas ao *QUIET*. Porém, reescrever as aplicações já existentes pode ser inviável.

Além disso, o foco deste trabalho reside em bases OLAP, o que desconsiderou o custo de manutenção das estruturas de índices (mediante as operações de update, insert e delete). Outro problema relevante consiste no fato desta abordagem não mencionar o acompanhamento dos índices reais, ou seja, uma vez materializado um índice não tem mais o seu benefício atualizado (incrementado).

4.5 COLT: Uma Ferramenta para a Criação Automática e Contínua de Índices

Em [41, 42] os autores apresentam um protótipo de um *framework* de auto-sintonia denominado COLT (*Continuos OnLine Tuning*), o qual monitora as consultas submetidas ao SGBD e ajusta de forma automática a configuração de índices, levando em consideração a restrição do espaço disponível para estas estruturas, com o objetivo de maximizar a performance das consultas. Este protótipo foi implementado de forma intrusiva no SGBD PostgreSQL.

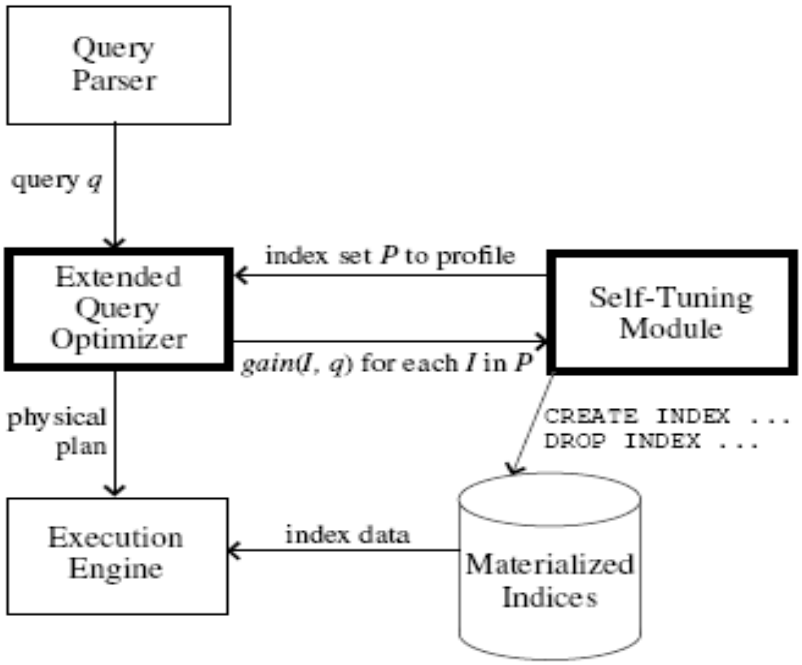

Figura 8: Arquitetura da Ferramenta COLT

Os autores apresentam como uma das principais contribuições desta proposta o fato

da ferramenta regular sua própria performance, reduzindo o seu *overhead* quando o SGBD está bem ajustado, e sendo mais agressivo quando ocorrem mudanças na carga de trabalho que implicam na necessidade de reajustar a configuração de índices do SGBD. Contudo, os autores não discutem como esta característica é atingida.

A Figura 8 mostra a arquitetura da ferramenta *COLT*. A implementação deste protótipo incluiu dois novos módulos no código do PostgreSQL: O otimizador de consultas estendido ("*Extended Query Optimizer*"), denominado EQO, e módulo de auto-sintonia ("*Self-Tuning Module*"), denominado STM.

4.5.1 O Otimizador de Consultas Estendido

Na implementação do protótipo *COLT* os autores substituíram o otimizador de consultas do PostgreSQL por um Otimizador de Consultas Estendido (EQO). A principal responsabilidade do EQO continua sendo a mesma do otimizador padrão do PostgreSQL: a seleção de um plano de execução ótimo para uma determinada consulta recebida como entrada. Contudo, o EQO estende as funcionalidades de um otimizador convencional adicionando a habilidade de tratar índices materializados e hipotéticos. Mais precisamente, dado uma consulta q e um índice i o EQO retorna a redução no custo da consulta q caso o índice i seja utilizado, ou seja, o ganho que o índice i pode proporcionar para o processamento da consulta q. Esta redução é denotada por $gain(i, q)$. Vale ressaltar que o ganho de um determinado índice i é sempre um valor não negativo, e $gain(i, q) = 0$ indica que a utilização do índice i não melhora o tempo de execução da consulta q.

O EQO calcula o ganho dos índices durante o processo de otimização da consulta corrente. Ou seja, a cada consulta recebida o EQO calcula o ganho para cada índice existente, seja ele materializado ou hipotético. Desta forma, quando o EQO recebe uma consulta q para ser otimizada ele primeiramente seleciona o melhor plano de execução para q utilizando o conjunto corrente de índices materializados.

Seja C_M o custo deste plano de execução inicial. Em seguida, o EQO recebe um conjunto de índices P. Para cada índice $i \in P$, o EQO computa um novo plano de execução para q, da seguinte maneira: (i) ie i é um índice materializado, o EQO gera o melhor plano de execução que não utiliza i. (ii) se i é um índice hipotético, o EQO gera o melhor plano de execução considerando a possibilidade de utilizar o índice i.

Os autores denominam esses planos de "*what-if plans*". Seja C_i o custo deste plano. O ganho do índice i para a consulta q ($\mathrm{g}ain(i, q)$) é calculado da seguinte forma:

$$\mathrm{g}ain(i, q) = |C_M - C_i|$$

4.5.2 O Módulo de Auto-Sintonia

O objetivo do módulo de alto-sintonia (STM) consiste em ajustar de forma automática e contínua a configuração de índices. Neste sentido, o STM tem duas tarefas principais:

- Para cada consulta q enviada ao SGBD, o STM seleciona um conjunto de índices candidatos P, ou seja, um conjunto de índices que potencialmente poderiam melhorar o desempenho do processamento da consulta q. O conjunto de índices P é enviado ao EQO para que este calcule o ganho de cada índice $i \in P$.

- Manter um conjunto de índices materializados, respeitando a restrição do espaço disponível, que maximize o desempenho da carga de trabalho submetida ao SGBD.

A fim de coordenar a execução destas duas tarefas, o STM divide as consultas recebidas em intervalos regulares denominados "época". Na implementação do COLT, uma época equivale à 10 (dez) consultas. Durante a duração de uma época o STM reune estatísticas acerca da carga de trabalho e obtém os ganhos dos índices candidatos. Ao final de uma época o conjunto de índices candidatos (materializados e hipotéticos) é analisado para se verificar a necessidade de alterações na configuração de índices corrente. Neste caso, índices hipotéticos podem vir a ser materializados e índices materializados podem ser removidos (voltando à situação de hipotéticos).

Os índices candidatos (materializados e hipotéticos) são classificados em três conjuntos distintos: M representa o conjunto dos índices materializados que são gerenciados pelo STM. O conjunto H ("*hot set*") contém os índices candidatos que foram relevantes para as consultas recentemente analisadas e que mostraram fortes evidências de que serão de grande utilidade para melhorar o desempenho da carga de trabalho. Já o conjunto C ("*cold set*") contém os índices candidatos que foram relevantes para as consultas recentemente analisadas, mas cuja utilidade para a carga de trabalho se mostrou apenas razoável.

A distribuição dos índices candidatos entre os conjuntos M, H e C é inicialmente guiada pelos valores dos seus ganhos. Observe que calcular o ganho de cada índice candidato para cada consulta processada é uma tarefa computacionalmente cara, pois calcular o ganho de um determinado índice i para uma determinada consulta q consiste em otimizar q mais uma vez. Assim, adicionou-se um limite para o número de otimizações realizadas em uma época. Contudo, este limite pode variar dependendo da estabilidade da carga de trabalho, sendo menor para cargas estáveis e maior para cargas em transição, por exemplo. Para atender essa restrição adotou-se a seguinte estratégia:

- Para os índices candidatos em C o STM calcula o ganho utilizando uma métrica simples baseada na seletividade dos predicados da consulta. Assim, não é necessário otimizar a consulta para calcular o ganho dos índices em C.

- Para os índices candidatos em M e H calcula-se o ganho normalmente (utilizando otimizações adicionais). Contudo, adotou-se a estratégia de calcular o ganho apenas de um sub-conjunto dos índices em M e H, os quais são escolhidos aleatoriamente. Logo, nem todos os índices em M e H terão seus ganhos calculados. Para estes índices, o STM estipula um valor para ganho baseado nos ganhos obtidos anteriormente em consultas "similares" à consulta corrente. Duas consultas são consideradas "similares" se envolvem as mesmas relações e seus predicados de seleção diferem apenas na seletividade. Caso um determinado índice não tenha obtido ganhos em consultas similares, o STM tentará calcular o seu ganho da forma padrão.

  Durante a etapa de seleção de índices candidatos para uma determinada consulta q, se um índice candidato ainda não existente é selecionado, este índice candidato é adicionado ao conjunto C. Ao final de uma época o STM utiliza os ganhos dos índices para redistribuí-los nos conjuntos M, H e C.

4.5.3 Comentários Gerais

Como esta abordagem pressupõe alterações no código do SGBD ela poderia ser classificada como um estratégia local e intrusiva. Além disso, o foco deste trabalho reside em bases OLAP, o que desconsiderou o custo de manutenção das estruturas de índices (mediante as operações de update, insert e delete). Outro problema relevante consiste no fato desta abordagem não mencionar o acompanhamento dos índices reais.

4.6 Manutenção Automática e Construção "On-the-fly" de Índices

Em [29] os autores apresentam uma abordagem para sintonia automática de índices, a qual baseia-se na utilização de índices hipotéticos (denominados *Soft Indexes*) e independe de qualquer interação do DBA. A abordagem proposta segue a linha das soluções intrusivas, sendo completamente integrada ao SGBD. A solução apresentada, monitora e coleta estatísticas continuamente acerca dos índices reais e hipotéticos e, com base nas informações coletadas, soluciona periodicamente o problema da seleção de índices (*Index Selection Problem - ISP*).

O aspecto dinâmico da sintonia automática de índices é descrito em [29] de acordo com o modelo genérico de sintonia automática, o qual consiste em um ciclo de Observação, Predição e Reação, como inicialmente proposto em [45]. Estas três fases são repetidas continuamente a fim de: monitorar o comportamento atual do sistema (cargas de trabalho e recomendações de índices), a partir das observações realizadas derivar uma decisão sobre comportamento futuro (alterações na configuração dos índices) e, se mudanças forem necessárias, aplicar estas mudanças durante a fase de reação. Vale ressaltar que a construção das estruturas de índices é integrada ao processador de consultas, o que possibilita a criação *online* (e *on-the-fly*) de índices através da utilização de dois novos operadores: *IndexBuildScan* e *SwitchPlan*. Estes novos operadores permitem, por exemplo, que a criação de um índice seja realizada em conjunto com uma operação *Table Scan*.

4.6.1 A Fase de Observação: Monitoramento da carga de trabalho e das estatísticas relativas aos índices candidatos

Primeiramente, cada consulta SQL capturada é analisada a fim de se descobrir índices candidatos. Para isso é utilizada a heurística de seleção de índices candidatos descrita em [28]. Em seguida, estima-se o tamanho de cada índice candidate selecionado. Nesta fase, cada consulta Q submetida ao SGBD é otimizada duas vezes, uma sem considerar índice algum, e outra considerando todos os índices candidatos.

Seja D o conjunto dos índices candidatos e I o conjunto dos índices utilizados durante a segunda otimização da consulta Q. Podemos afirmar que I é um subconjunto de D ($I \subseteq D$).

O benefício do conjunto de índices I, em uma determinada consulta Q, pode ser calculado com base no custo de execução de Q nas duas otimizações anteriores, ou seja, sem considerar a existência de índices ($cost(Q)$) e considerando a existência dos índices candidatos ($cost(Q, I)$).

$$profit(Q, I) = cost(Q) - cost(Q, I)$$

Logicamente, os índices que pertencem ao conjunto I podem contribuir de forma diferente para o benefício total. Em [40] os autores avaliam diferentes abordagens para atribuir a cada índice $i \in I$ parte do benefício total de forma justa (proporcional). Baseado nestes estudos, os autores sugeriram a seguinte aproximação:

$$profit(Q, I) = \frac{profit(Q, I) \times size(I)}{\sum_{I_j \in I} size(I_j)}$$

Além disso, um comando SQL de atualização (*update*) pode implicar na necessidade de se atualizar as estruturas de índices. Neste caso, o custo de atualização da estrutura de índice é considerado como um "malefício" (ou seja, um benefício negativo).

Assim, a estimativa do custo de uma operação de atualização Q_U deve considerar o número de linhas afetadas por Q_U, a altura da árvore e um fator F derivado empiricamente para representar o custo de atualização de uma entrada na estrutura de índices.

$$profit(Q_U, I) = -height(I) \times nrows(Q_U) \times F$$

Ao se atualizar as estatísticas dos índices candidatos deve-se considerar que as definições de determinados índices podem se sobrepor. Assim, uma consulta que utiliza um índice definido sobre o atributo $R(A)$ também poderia utilizar um outro índice definido sobre os atributos $R(A, B)$. Logo, o benefício do índice definido sobre $R(A)$ também deveria ser atribuído ao índice definido sobre $R(A, B)$.

Desta forma, dados dois índices I_1 e I_2, tais que $I_1 \in D$ e $I_2 \in D$, diz-se que I_1 está contido em I_2 ($I_1 \sqsubset I_2$) se os atributos do índice I_1 são um prefixo de I_2. Neste caso, o benefício deve ser atribuído aos dois índices.

$$\forall I_i \in D : I_r \sqsubset I_i \Rightarrow profit(Q, I_i) = profit(Q, I_r)$$

Para lidar com o aspecto temporal da abordagem proposta os autores utilizam o conceito de época [40], o qual simplesmente delimita a duração de um período de observação. A idéia de "época" representa uma alternativa simples para tratar dois problemas: a) a desatualização das estatísticas e b) a escolha do momento de se iniciar a fase de predição. Estatísticas coletadas remotamente podem não mais representar de forma satisfatória o estado atual do sistema. A utilização de estatísticas desatualizados no processo de predição de índices pode resultar em alterações de configuração que levem a perda de desempenho. A escolha do momento de se iniciar a fase de predição também é uma questão relevante. Executar a fase de predição sempre que uma consulta for submetida ao SGBD (como proposto em [40, 37, 38, 16, 33, 27]) pode gerar uma sobrecarga desnecessária e resultar em configurações instáveis. Neste sentido a utilização do conceito de época pode levar a uma baixa sobrecarga e à configurações mais estáveis.

A duração de uma época pode ser definida em termos de tempo, número de consultas, valor do benefício de um índice candidato, ou ainda, como utilizado na implementação descrita em [29], do número máximo de recomendações para um mesmo índice.

Além disso, esta abordagem sugere dar um peso maior para os benefícios gerados pelas recomendações mais recentes. Assim, os benefícios calculados mais recentemente terão um peso maior que os benefícios calculados mais remotamente. Esta característica pode ser implementada da seguinte forma: para cada recomendação, ou seja, para cada benefício calculado, atribui-se um marcador de tempo (*timestamp*) monotonicamente crescente, $ts_1, ts_2, ...ts_k$. Assim, uma recomendação com *timestamp* ts_2 é posterior a uma recomendação com *timestamp* ts_1. Seja ts_E *timestamp* referente ao encerramento de uma época, o benefício de um determinado índice I pode ser calculado da seguinte forma:

$$benefit(I) = \sum_{j=1}^{k} \frac{profit(I, ts_j)}{ts_E - ts_j}$$

Neste sentido, o benefício e o *timestamp* de cada recomendação são armazenados em campos de tamanho fixo. Uma época termina quando o valor do *timestamp* gerado para uma determinada recomendação supera o tamanho utilizado para este campo.

4.6.2 A Fase de Predição: Seleção dinâmica de índices candidatos

O encerramento de uma época dispara o início da fase de decisão, a qual consiste em solucionar o problema da seleção de índices (ISP) usando as informações obtidas durante a fase de observação. Para cada índice I utiliza-se: o benefício de I ($benefit(I)$), o tamanho estimado de I ($size(I)$) e o estado atual de I ($state(I)$), onde este último indica se o índice é materializado ou não ($soft$). Além disso, $plimit$ representa a restrição de espaço para a materialização de índices. Para solucionar o problema em questão (ISP), a abordagem utiliza um algoritmo guloso como mostrado na Figura 9. Este algoritmo utiliza como entrada o conjunto de todos os índices (materializados ou hipotéticos) ordenados pelo benefício relativo, o qual pode ser calculado como segue:

$$relative\_benefit(I) = \frac{benefit(I)}{size(I)}$$

Assim, uma nova configuração de índices $\bar{C}$ é calculada utilizando-se o algoritmo da Figura 9. A fim de evitar a freqüente inserção/remoção dos mesmos índices, uma nova configuração $\bar{C}$ somente irá substituir a configuração atual C, se o benefício de $\bar{C}$ exceder o benefício da configuração atual multiplicado por um fator F, o qual usualmente é um valor constante < 1.0.

A materialização de uma nova configuração implica na criação de novos índices, os quais são considerados índices adiados (*deferred indexes*), ou seja, índices vazios que serão povoados posteriormente, uma vez que a criação imediata destas estruturas podem degradar o desempenho do sistema, e remoção daqueles índices que se mostraram de pouca utilidade. Ainda assim, um índice removido continua existindo no catálogo do sistema de banco de dados como um índice virtual (*Soft*), podendo receber benefícios e futuramente ser materializado novamente. O custo de exclusão de um índice (benefício negativo) é ignorado, uma vez que os autores assumem que este seria extremamente barato.

A complexidade do algoritmo guloso é $O(nlogn)$ para n índices candidatos, esta complexidade deve-se ao esforço necessário para a ordenação da lista dos índices. O esforço

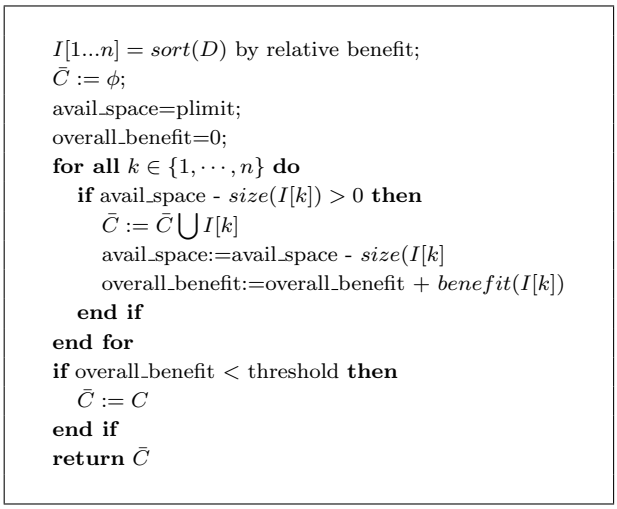

Figura 9: Algoritmo Guloso para Seleção de Índices.

total é razoável, uma vez que o espaço de busca utilizado durante a segunda otimização é limitado. Logicamente, esta abordagem não assegura uma solução ótima, mas os autores argumentam que o resultado é suficientemente preciso, principalmente se considerarmos que diversas variáveis de entrada são estimadas e que o resultado é utilizado como uma previsão do uso futuro dos índices.

4.6.3 A Fase de Reação: Criação de lindices *on-the-fly* (*online*)

Caso, durante a fase anterior, seja detectado a necessidade de se alterar a configuração de índices, os novos índices são criados como adiados, ou seja, estão prontos para posterior materialização. Em geral, a construção do índice pode ser realizada a qualquer tempo. Uma possibilidade interessante seria utilizar momentos de baixa atividade do sistema de banco de dados. Além disso, os autores investigaram os ganhos de desempenho obtidos através da integração da criação de índices com o processamento de consultas, fazendo, por exemplo, a construção de um índice durante a execução de um Table Scan, como proposto em [21]. Neste sentido, foram introduzidos dois novos operadores IndexBuildScan e SwitchPlan. O operador IndexBuildScan estende a operação Table Scan, sobre uma tabela t, a fim de criar um ou mais índices adiados. Este operador recebe como parâmetro a lista de índices adiados L definidos sobre a tabela t, os quais serão construídos durante a varredura de t.

4.6.4 Comentários Gerais

As idéias propostas foram implementadas e avaliadas como uma extensão do SGBD PostgreSQL 7.4. A avaliação de desempenho foi realizada utilizando uma carga TPC-H. Contudo, a abordagem proposta em [29] limita-se à manutenção de índices secundários, restringiu-se à utilização de índices implementados através de árvores B^+ e foi implementada a partir de alterações e extensões realizadas junto ao código do PostgreSQL versão 7.3, de forma intrusiva. Logo, ao surgir uma nova versão do SGBD PostgreSQL será

necessário refazer todas estas alterações. Assim, essa abordagem pode ser classificada como local e intrusiva.

Além disso, vale ressaltar que o custo de criação de um índice não pode ser eliminado completamente. Os experimentos realizados em [29] mostram que mesmo a abordagem da construção *on-the-fly* de índices resulta em esperas no processamento de consultas, tais demoras podem ser inaceitáveis se as aplicações necessitarem de garantias quanto ao tempo de resposta. Uma possível solução seria construir somente partes do índice durante o processamento de uma determinada consulta. Desta forma, o custo de criação do índice seria distribuído entre várias consultas [29, 23, 22].

4.7 AutoAdmin Online: Uma Ferramenta para Ajuste Online do Projeto Físico

Em [7, 8] os autores apresentam uma ferramenta de sintonia automática de índices que foi implementada como uma extensão do Microsoft SQL Server. Este novo componente executa continuamente e, reagindo a variações na carga de trabalho ou nas características dos dados, modifica de forma automática o projeto físico do banco de dados. Os algoritmos propostos apresentam baixa sobrecarga e levam em consideração restrições no espaço de armazenamento, o custo da atualização das estruturas de índices causadas pelos operações de *update* e o custo de criar estruturas físicas temporárias. A Figura 10 ilustra os componentes do SQL Server que foram modificados ou adicionados a fim de possibilitar a sintonia automática do projeto físico. O gerenciador de metadados foi estendido com a finalidade de fornecer suporte para a implementação do conceito de índices candidatos hipotéticos (os quais não são materializados e, portanto, não podem ser utilizados durante o processamento normal das consultas). Na representação interna dos índices (tanto reais quanto hipotéticos) foi adicionado um pequeno conjunto de contadores, os quais representam o benefício de cada índice para a carga de trabalho. Esse benefício é utilizado na seleção dos índices que devem ser fisicamente criados ou removidos.

Inicialmente, durante o processo de otimização de uma consulta, a chamada ao otimizador é rapidamente desviada para o módulo de análise de índices (*Index analysis*) a fim de se identificar um conjunto de índices candidatos relevantes (ou seja, que podem melhorar o desempenho) para a consulta que está sendo otimizada. Com esta finalidade, utiliza-se uma árvore de requisição *AND/OR* e transformações locais. Após esta análise os contadores são atualizados. Após este desvio a consulta é otimizada normalmente. Em seguida, durante a execução da consulta, utiliza-se o pré-processamento realizado na fase de otimização e calcula-se o benefício potencial que foi perdido pela não materialização dos índices hipotéticos. Além disso, calcula-se também a utilidade dos índices existentes (reais), que foram utilizados durante a execução da consulta. Essas estimativas são realizadas pelo módulo "*Cost/Benefit Adustment*" (Figura 10). Vale ressaltar que estes cálculos são executados sem que seja necessário otimizar a consulta uma segunda vez.

Após a execução de cada consulta (ou após a execução de um determinado número de consultas, caso se deseje diminuir a sobrecarga do processo de sintonia automática) analisa-se a relação custo/benefício dos índices candidatos e a possibilidade de melhoria no desempenho da carga de trabalho mediante a criação ou remoção de índices. Por exemplo, se o benefício de um determinado índice em um instante futuro for maior que o seu custo de criação (e não for negativo), isto indica que o índice deve ser criado para o período de tempo em questão. Caso alguma alteração no projeto físico se mostrar

benéfica as devidas requisições de criação e/ou remoção de índices são enviadas ao módulo de gerenciamento assíncrono (*Asynchronous Task Manager*), o qual irá executar os comandos DDL necessários. Adicionalmente, se existir uma restrição quanto à capacidade de armazenamento, impedindo que todos os índices selecionados sejam criados, a ferramenta deve decidir: que índices materializar, se a eliminação de um índice existente poderia liberar espaço para dois ou mais outros índices cuja soma dos benefícios seja maior que a do índice eliminado, se é possível juntar ("*merge*") dois ou mais índices. Contudo, os DBAs podem monitorar o estado interno da ferramenta utilizando uma aplicação cliente. O protótipo implementado foi avaliado utilizando-se uma carga de trabalho TPC-H.

A natureza dinâmica ("*online*") do problema em questão implica que a solução encontrada, a qual baseia-se nas consultas previamente executadas, geralmente, estará atrasada em relação à solução ótima (que busca melhorar o desempenho das consultas futuras). Contudo, os autores afirmam ter encontrado evidências que as decisões tomadas pela solução proposta não sofre perdas significantes com este atraso.

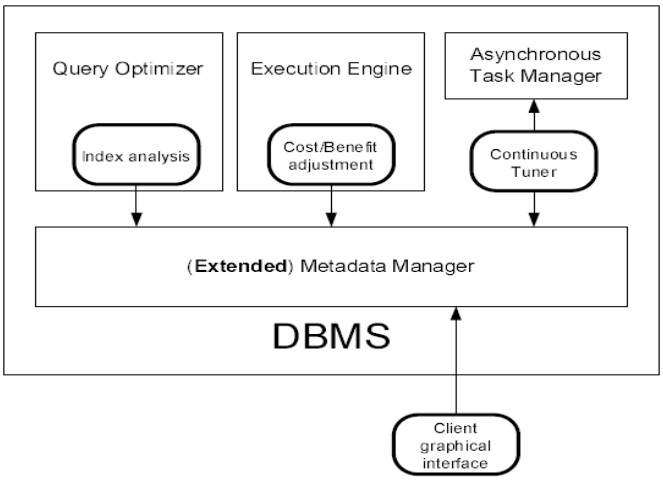

Figura 10: Arquitetura da Ferramenta *AutoAdmin Online*

4.7.1 Comentários Gerais

As idéias propostas foram implementadas e avaliadas como uma extensão do SQL Server 2005. A avaliação de desempenho foi realizada utilizando uma carga TPC-H. Contudo, esta abordagem não investigou o problema de recriação automática das estruturas de índices ("*reindex*"). Além disso, restringe-se a manutenção de índices secundários e organizados exclusivamente em árvore B^+

4.8 Discussão

A grande maioria das soluções encontradas na literatura e discutidas até aqui, embora apresentem bons resultados em seus experimentos, não contemplam com a devida importância alguns aspectos relevantes para a manutenção automática do projeto físico de bancos de dados. Estas abordagens podem ser classificadas como locais e intrusivas, restringem-se à manutenção de índices secundários organizados exclusivamente em árvore

B^+ e não avaliam os benefícios obtidos através da pré-criação de índices envolvidos em chaves primárias e estrangeiras. Além disso, tais abordagens apresentam excessiva sobrecarga ("*overhead*").

A fim de realizar uma análise comparativa mais detalhada entre as soluções existentes, listamos um conjunto de características que serão utilizadas para diferenciar as abordagens estudadas:

- Estruturas de acesso utilizadas (C1);
- Ações de gerenciamento executadas sobre as estruturas de acesso utilizadas (C2);
- Nível de acoplamento da solução ao SGBD utilizado (C3);
- SGBDs suportados (C4);
- Quantidade de heurísticas utilizadas (C5);
- Quantidade de índices hipotéticos gerados (C6);
- Modelo de custos utilizado (C7);
- Consideração de restrições de armazenamento (C8);
- Atribuição proporcional de benefícios (C9);
- Número de vezes que cada consulta é otimizada (C10);
- Quantidade de comandos criados ou alterados junto ao código fonte do SGBD utilizado (C11);
- *Benchmark* utilizado na avaliação da solução (C12);
- Consideração da estabilidade da configuração de índices (C13);
- Utilização do conceito de generalidade (C14);
- Utilização do conceito de relevância (C15);
- Avaliação da pré-criação de índices envolvidos em chaves primárias e estrangeiras (C16);

A Tabela 1 apresenta, de forma sumarizada, uma comparação entre as soluções existentes. Esta comparação baseia-se nas características previamente listadas [2].

[2]* Os autores não informaram ou não deixaram claro em suas publicações.

Característica	Sal04	Morelli06	Sattler03	Sch06	Luh07	Bruno07
C1	Índice Sec., B^+	Índice Sec., B^+	Índice Sec., B^+	Índice Sec., B^+	Índice Sec., B^+	Índice Sec., B^+
C2	Criação, Remoção	Criação, Remoção, Reorganização	Criação	Criação, Remoção	Criação, Remoção	Criação, Remoção
C3	Intrusivo	Intrusivo	Intrusivo	Intrusivo	Intrusivo	Intrusivo
C4	PostgreSQL	PostgreSQL	DB2	PostgreSQL	PostgreSQL	SQL Server
C5	2	2	2	2	2	1
C6	Alta	Alta	Alta	Alta	Alta	Baixa
C7	Interno	Interno	Interno	Interno	Interno	Interno
C8	Não	Não	Sim	Sim	Sim	Sim
C9	Não	Parcial	Não	Sim	Sim	Sim
C10	3	3	2	2	2	1
C11	4	6	0	*	*	*
C12	TPC-C	TPC-H	TPC-H	*	TPC-H	TPC-H
C13	Não	Não	Sim	Sim	Sim	Sim
C14	Não	Não	Não	Não	Não	Não
C15	Não	Não	Não	Não	Não	Não
C16	Não	Não	Não	Não	Não	Não

Tabela 1: Análise Comparativa das Abordagens para Manutenção Automática de Índices.

5 Conclusões

O desempenho dos servidores de bancos de dados é fator chave para o sucesso das aplicações de missão-crítica. Para estas aplicações, uma baixa *performance* significa perdas de receita e de oportunidades de negócio. A fim de assegurar um desempenho sempre aceitável torna-se necessário monitorar continuamente a infra-estrutura dos servidores de bancos de dados, e, em caso de eventos inesperados que possam comprometer a *performance* do sistema, deve-se reagir de forma imediata, solucionando-se os problemas encontrados no menor espaço de tempo possível, com rapidez e eficiência. Neste artigo, apresentamos uma visão geral sobre auto-sintonia do projeto físico de banco de dados. Além disso, este trabalho propõe uma nova taxonomia para a classificação das pesquisas em auto-sintonia e realiza uma análise comparativa detalhada entre as principais propostas, encontradas na literatura.

Referências

[1] S. Agrawal, S. Chaudhuri, K. L., M. A., V. Narasayya, and M. Syamala. Database tuning advisor for microsoft sql server 2005. In *In Proceedings of the International Conference on Very Large Databases (VLDB)*, 2004.

[2] S. Agrawal, S. Chaudhuri, and V. Narasayya. Automated selection of materialized views and indexes for sql databases. In *In Proceedings of VLDB*, pages 496–505, 2000.

[3] S. Agrawal, E. Chu, and V. Narasayya. Automated physical design tuning: Workload as a sequence. In *Proceedings of the ACM SIGMOD*, 2006.

[4] N. Bruno and S. Chaudhuri. Automatic physical database tuning: a relaxation-based approach. In *Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data*, pages 227–238, New York, NY, USA, 2005. ACM.

[5] N. Bruno and S. Chaudhuri. Physical design refinement: The merge-reduce approach. In *Proceedings of the EDBT (EDBT06)*, 2006.

[6] N. Bruno and S. Chaudhuri. To tune or not to tune? a lightweight physical design alerter. In *Proceedings of the 32rd International Conference on Very Large Databases (VLDB06)*, 2006.

[7] N. Bruno and S. Chaudhuri. An online approach to physical design tuning. In *Proceedings of the ICDE Conference (ICDE07)*, 2007.

[8] N. Bruno and S. Chaudhuri. Online autoadmin: (physical design tuning). In *Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data (SIGMOD07)*, pages 1067–1069. ACM, 2007.

[9] D. Burleson. *Creating a Self-Tuning Oracle Database*. Rampant, 2004.

[10] S. Chaudhuri, M. Datar, and V. Narasayya. Index selection for databases: A hardness study and a principled heuristic solution. *IEEE Transactions on Knowledge and Data Engineering*, 16(11):1313–1323, 2004.

[11] S. Chaudhuri and V. Narasayya. An efficient, cost-driven index selection tool for microsoft sql server. In *In Proceedings of VLDB*, pages 146–155, 1997.

[12] S. Chaudhuri and V. Narasayya. Autoadmin "what-if" index analysis utility. In *In Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 367–377, 1998.

[13] S. Chaudhuri and V. Narasayya. Microsoft index tuning wizard for sql server 7.0. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 553–554, 1998.

[14] S. Chaudhuri and V. Narasayya. Self-tuning database systems: A decade of progress. In *Proceedings of the 33rd International Conference on Very Large Databases (VLDB07)*, 2007.

[15] R. L. C. Costa and S. Lifschitz. Index self-tuning and agent-based databases. In *In Proceedings of the Latin-American Conference on Informatics (CLEI)*, 2002.

[16] R. L. C. Costa, S. Lifschitz, M. Noronha, and M. V. Salles. Implementation of an agent architecture for automated index tuning. In *Proceedings of the ICDE Workshops*, 2005.

[17] B. Dageville, D. Das, K. Dias, K. Yagoub, M. Zait, and M. Ziauddin. Automatic sql tuning in oracle 10g. In *In Proceedings of the International Conference on Very Large Databases (VLDB)*, pages 1098–1109, 2004.

[18] M. R. Database Group. Autoadmin project. .

[19] K. Dias, M. Ramacher, U. Shaft, V. Venkataramani, and G. Wood. Automatic performance diagnosis and tuning in oracle. In *Proceedings of the CIDR Conference*, pages 84–94, 2005.

[20] L. G. and S. Lightstone. Smart: Making db2 (more) autonomic. In *In Proceedings of the International Conference on Very Large Databases (VLDB)*, 2002.

[21] G. Graefe. Dynamic query evaluation plans: Some course corrections? *IEEE Data Eng. Bull.*, 23(2):3–6, 2000.

[22] S. Idreos, M. L. Kersten, and S. Manegold. Database cracking. In *Proceedings of the Third Biennial Conference on Innovative Data Systems Research (CIDR07)*, pages 68–78, 2007.

[23] S. Idreos, M. L. Kersten, and S. Manegold. Updating a cracked database. In *Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data (SIGMOD07)*, pages 413–424, New York, NY, USA, 2007. ACM.

[24] E. Kendall, P. Krishna, C. Pathak, and C. Suresh. *A Framework for Agent Systems. In Implementing Application Frameworks - Object-Oriented Frameworks at Work, M. Fayad et al. (eds.).* John Wiley & Sons, 1999.

[25] A. C. Konig and S. U. Nabar. Scalable exploration of physical database design. In *Proceedings of the 22nd International Conference on Data Engineering (ICDE'06)*, page 37, Washington, DC, USA, 2006. IEEE Computer Society.

[26] S. Lifschitz, A. Y. M. Milanes, and M. V. Salles. State of the art in self-tuning relational database systems (in portuguese). Technical report, Departamento de Informática,PUC-Rio, 2004.

[27] S. Lifschitz and E. T. Morelli. Towards autonomic index maintenance. In *Proceedings of the Brazilian Symposium on Database*, 2006.

[28] G. Lohman, G. Valentin, D. Zilio, M. Zuliani, and A. Skelley. Db2 advisor: An optimizer smart enough to recommend its own indexes. In *In Proceedings of the IEEE International Conference on Data Engineering (ICDE)*, pages 101–110, 2000.

[29] M. Luhring, K. Sattler, K. Schmidt, and E. Schallehn. Autonomous management of soft indexes. In *Proceedings of the 23rd IEEE International Conference on Data Engineering Workshop (ICDE'07)*, 2007.

[30] A. Milanés. *Uma arquitetura para auto-sintonia global de SGBDs usando agentes.* PhD thesis, Department of Informatics, PUC-Rio, 2004.

[31] A. Milanés and S. Lifschitz. Design and implementation of a global self-tuning architecture. In *Proceedings of the Brazilian Symposium on Database*, 2005.

[32] J. M. Monteiro, S. Lifschitz, and A. Brayner. An architecture for automated index tuning. In *In Ph.D. and M.S. Workshop of the Brazilian Symposium on Database*, 2006.

[33] E. M. T. Morelli. *Recriação Automática de Índices em um SGBD Relacional (in portuguese)*. PhD thesis, Department of Informatics, PUC-Rio, 2006.

[34] Osdl. .

[35] S. Papadomanolakis and A. Ailamaki. An integer linear programming approach to database design. *IEEE International Conference on Data Engineering*, pages 442–449, 2007.

[36] Postgresql. .

[37] M. V. Salles. *Autonomic index creation in databases (in portuguese)*. PhD thesis, Department of Informatics, PUC-Rio, 2004.

[38] M. V. Salles and S. Lifschitz. Autonomic index management. In *In Proceedings of the International Conference on Autonomic Computing (ICAC)*, 2005.

[39] K. Sattler, I. Geist, and E. Schallehn. Quiet: Continuous query-driven index tuning. In *Proceedings of the 29th international conference on Very large data bases (VLDB03)*, pages 1129–1132. VLDB Endowment, 2003.

[40] K.-U. Sattler, E. Schallehn, and I. Geist. Autonomous query-driven index tuning. In *IDEAS '04: Proceedings of the International Database Engineering and Applications Symposium (IDEAS'04)*, pages 439–448, Washington, DC, USA, 2004. IEEE Computer Society.

[41] K. Schnaitter, S. Abiteboul, T. Milo, and N. Polyzotis. Colt: continuous on-line tuning. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD06)*, pages 793–795, New York, NY, USA, 2006. ACM.

[42] K. Schnaitter, S. Abiteboul, T. Milo, and N. Polyzotis. On-line index selection for shifting workloads. In *2nd International Workshop on Self-Managing Database Systems (SMDB 2007)*, pages 459–468, 2007.

[43] D. Shasha and P. Bonnet. *Database Tuning: Principles, Experiments and Troubleshooting Techniques*. Morgan Kaufmann, 2003.

[44] Tpc. .

[45] G. Weikum, A. Hasse, C. Monkeberg, and P. Zabback. The COMFORT automatic tuning project, invited project review. *Information Systems*, 19(5):381–432, 1994.

[46] G. Weikum, A. Mönkeberg, C. Hasse, and P. Zabback. Self-tuning database technology and information services: from wishful thinking to viable engineering. In *In Proceedings of the International Conference on Very Large Databases (VLDB)*, pages 20–31, 2002.

[47] E. Yourdon and L. L. Constantine. *Projeto Estruturado de Sistemas*. Campus, 1990.

[48] D. Zilio, J. Rao, S. Lightstone, G. Lohman, A. Storm, C. Garcia-Arellano, and S. Fadden. Db2 design advisor: Integrated automatic physical database design. In *In Proceedings of the International Conference on Very Large Databases (VLDB)*, pages 1087–1097, 2004.