3
Trabalhos Relacionados

A utilização de bancos de dados em escala crescente, o aumento vertiginoso da quantidade de informações armazenadas e a alta complexidade das aplicações atuais tem impulsionado o desenvolvimento de SGBDs capazes de alta sintonia (Horn01). Idealmente, o que se deseja é obter os benefícios do uso de SGBDs sem que seja necessário arcar com os custos relativos à sua administração e ao gerenciamento de seu desempenho. Este é um objetivo audacioso. Desta forma, diversos trabalhos têm buscado encontrar soluções automáticas para diferentes aspectos envolvidos na sintonia de SGBDs, procurando reduzir, na medida do possível, os custos de seu gerenciamento.

Este capítulo tem por objetivo apresentar o estado da arte da área de sintonia automática do projeto físico de bancos de dados. As propostas que serão discutidas procuram reduzir as intervenções humanas nas tarefas de ajustes de desempenho. Primeiramente discutiremos uma classificação das pesquisas anteriormente realizadas na área de sintonia automática (3.1). Em seguida, apresentaremos as abordagens relacionadas à auto-sintonia global (3.2) e à auto-sintonia local (3.3). Ainda na seção 3.3 discutiremos as principais propostas para solucionar um dos problemas mais relevantes na área de auto-sintonia local: a manutenção automática das estruturas de índices.

3.1
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 (Lif04a). 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. Os trabalhos de pesquisa existentes foram classificados em dois grandes grupos: Auto-Sintonia Global e Auto-Sintonia Local.

No primeiro grupo estão os trabalhos de auto-sintonia global, os quais procuram estudar como é possível tomar ações de sintonia que tragam benefícios de desempenho para o sistema como um todo. Este tipo de abor-

dagem procura encontrar um equilíbrio entre as diversas considerações locais de sintonia de forma a alcançar um desempenho global que seja melhor do 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 (Weikum02).

No segundo grupo estão os trabalhos de auto-sintonia local, que procuram 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.

Contudo, devido ao grande número de trabalhos publicados nos últimos anos e a grande variedade de ferramentas, estratégias e abordagens propostas, faz-se necessário uma nova forma de classificar as linhas de pesquisa existentes na área de sintonia automática. Neste trabalho, é proposta uma nova classificação baseada em duas dimensões ou eixos: (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 3.1.

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 (Yourdon90), 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 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, sendo, portanto, 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. Para ilustrar este fato, considere a implementação realizada em (Cos05, Sal04) onde, somente para codificação das características relacionadas à existência de índices hipotéticos no PostgreSQL 7.4, foram alterados 38 arquivos diferentes e mais de 169 linhas de código.

Figura 3.1: Classificação das Pesquisas em Auto-Sintonia

3.2 Auto-Sintonia Global de SGBDs

Em (Milanes05, Milanes04) 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 Multiobjetivo, 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 (Morelli06a).

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 (Milanes04) 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 (kendall99), 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. 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 (Dag04). Nessa versão foi incluído um componente especializado em diagnóstico automático. Este componente considera em suas análises o "*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.

3.3 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. No contexto do projeto AutoAdmin (AutoAdmin) da *Microsoft*, iniciado em 1997, Chaudhuri e Narasayya, em (Cha97, Cha98a, Cha98b), 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 (Cha98b), os autores utilizam uma técnica de otimização denominada "*hill climbing*" (referenciada no artigo como um algoritmo guloso), a qual pertence à uma família de estratégias de buscas locais, como estratégia de busca. Trabalhos adicionais sobre as heurísticas de seleção de índices, realizados pelos mesmo autores, podem ser encontrados em (Cha04). Dois anos depois, os trabalhos evoluíram para a seleção automática, não apenas de índices, mas também de visões materializadas (Agra00). Acompanhando o lançamento da versão 2005 do *SQL Server*, o processo foi estendido para que também fossem sugeridos particionamentos em grandes tabelas (Agra04).

Em (Cha04) 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. Destaque-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 campos 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 (Bruno05). Neste trabalho, os autores propõem que a etapa de seleção de índices candidatos (que inclui a enumeração e o "*merging*" de índices candidatos sejam realizadas durante e em conjunto com o processo de otimização da consulta, e não mais sejam realizados separadamente e por heurísticas distintas. A principal vantagem dessa abordagem é a diminuição do número de estruturas candidatas. Além disso, 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 essas novas configurações, as heurísticas propostas podem remover ou juntar um subconjunto 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 (Papa07) 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 (Konig06), os autores propõem uma forma alternativa de análise de cargas de trabalho, baseado em amostragens, com o objetivo de diminuir o "*overhead*" gasto neste processo.

O modelo de sintonia utilizado nos trabalhos anteriores assume que uma carga de trabalho consiste em um conjunto de consultas e atualizações. Contudo, (Agra06) 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 uma abordagem ótima para a

sintonia de seqüências, a qual baseia-se no problema do caminho mínimo, foi apresentada, além de um algoritmo guloso, que se mostrou bastante eficiente na prática. As técnicas propostas naquele trabalho foram implementadas e avaliadas através de uma extensão do *Database Tuning Advisor*, o qual faz parte do *Microsoft SQL Server 2005*.

Já (Bruno06a) 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 não poderiam ser utilizadas, uma vez que esta amostra é um pré-requisito para o funcionamento de tais ferramentas. O DBA deve estar apto a fornecer um projeto físico inicial, baseado em seu conhecimento. Porém, esta configuração inicial pode violar importantes restrições, como, por exemplo, o espaço disponível para armazenamento. A abordagem proposta em (Bruno06a) parte de uma configuração inicial fornecida pelo DBA e progressivamente refina esta solução até chegar em uma nova solução que satisfaça as restrições existente (como o espaço disponível, por exemplo). Esse 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.

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 amostra 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 sofrendo 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 (Bruno06b) 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 apresentado neste trabalho foi divido em dois componentes: um cliente, escrito em C++ e outro servidor, implementado junto ao código do SQL Server 2005.

Em (Cha07a), 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" (Idreos07a, Idreos07b). 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. Além disso, essas pesquisas contemplaram a criação e remoção automática de índices, visões materializadas e índices definidos sobre visões materializadas, além do particionamento de grandes tabelas. Contudo, nenhuma dessas iniciativas investigou o problema de recriação automática das estruturas de índices ("*reindex*").

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 (Loh02), que visa enriquecer o gerenciador DB2 com características autonômicas. Em (Loh00), Lohman et al. descreve 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, índices hipotéticos, para todas as colunas relevantes, são gerados. 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 (hipotéticos) apresentada neste trabalho foi implementada em (Sal04, Cos05, Sal05, Morelli06a, Lif06). Já em (Lightstone02), os autores afirmam que o protótipo implementado realiza a reorganização automática das estruturas de índice, mas não discute como esta funcionalidade é alcançada. Finalmente, Lohman et al. (Zilio04) 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. Essas pesquisas contemplaram a criação e remoção automática de índices, visões materializadas e índices definidos sobre visões materializadas. Contudo, nenhuma dessas iniciativas investigou o problema de recriação automática das estruturas de índices.

A Oracle também tem investido bastante nos últimos anos buscando adicionar funcionalidades de auto-sintonia em seu SGBD. A última versão

do seu SGBD, Oracle 10g, oferece vários recursos que permitem a criação de *scripts* contendo instruções que dinamicamente ajustem o ambiente, segundo os níveis de utilização (Burleson04). Já em (Dias05, Dageville06) 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 (Morelli06a). Estas iniciativas 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 (PostgreSQL). Esta ferramenta, periodicamente, compacta tabelas eliminando fisicamente *tuplas* que tenham sido marcadas para exclusão (Morelli06a).

Vale destacar ainda que Shasha demonstrou, através de testes práticos, que a não manutenção periódica de índices faz com que o desempenho seja degradado com o tempo (Sha03). Contudo, poucos trabalhos procuraram solucionar o problema da recriação automática de índices.

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. Uma 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 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.

Recentemente, alguns trabalhos têm sido propostos na direção de buscar uma solução completa para a manutenção automática do projeto físico. Alguns trabalhos procuram estender o PostgreSQL, através da geração de protótipos, adicionando funcionalidades de auto-sintonia (Sal04, Cos05, Sal05, Morelli06a, Lif06, Sch06, Luh07). A Microsoft também tem investido na tentativa de estender o SQL Server 2005 adicionando funcionalidades que possibilite a manutenção automática do projeto físico (Bruno07a, Bruno07b).

Já em (Sattler03, Kai04) 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 como, por exemplo, o PostgreSQL e o SQL Server 2005. 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. Portanto, todas estas soluções apresentam uma abordagem local e intrusiva. A seguir, estas abordagens serão discutidas com maiores detalhes.

3.3.1 Criação e Remoção Automática de Índices

O Trabalho proposto em (Sal04, Sal05, Cos05) apresenta um "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 para 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 (integrada) 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.

Idéia Básica

A idéia básica daquela proposta consiste em utilizar duas heurísticas: uma para seleção de índices candidatos e outro para a seleção final de índices. A seguir, descrevemos brevemente essas duas heurísticas.

A primeira heurística ocupa-se em analisar as cláusulas SQL submetidas ao SGBD, e enumerar índices que potencialmente melhorariam esta cláusula. Esta heurística, adaptada de (Loh00), 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, ou seja, índices que existem apenas na metabase do SGBD, mas que não existem fisicamente. Como a heurística de seleção de índices candidatos presume a existência de índices do tipo "*what-if*" (hipotéticos), tais como propostos em (Cha98a), houve a necessidade de estender a funcionalidade do SGBD PostgreSQL para que o otimizador levasse em conta a presença de índices hipotéticos (Sal05). Comandos foram estendidos para que tais índices virtuais pudessem ser criados ou destruídos, e também para que aparecessem em planos de execução. Em (Sal05), os autores mostram que a utilização deste tipo de índice não causou sobrecarga ao SGBD, mesmo quando criados em tabelas volumosas.

Já a segunda heurística, denominada Heurística de Benefícios, procura decidir de forma "*on-the-fly*", 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. Uma vez que se obtém o custo de cada comando e quanto este custo poderia ser reduzido, caso existissem determinados índices, pode-se calcular o benefícios desses índices para o comando em questão. 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 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 (Cos02), porém nunca tinha sido testada para uso efetivo em um SGBD. Em (Sal04) propõem-se diversas extensões ao trabalho apresentado em (Cos02), tais como a definição de equações para cálculos dos benefícios, e refinamento das estratégias de avaliação de consultas, que tornaram a heurística mais realista.

Em (Sal04) propõe-se 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, quando usado em um comando com um índice com significativa contribuição, terá um benefício acumulado falacioso (Morelli06a).

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. Segundo (Sal04), 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 (Morelli06a).

Uma alternativa mais justa consiste 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 (Sal04), seria concedido um valor variável que dependeria da real contribuição do índice ao comando no qual estaria participando (Morelli06a).

(Cha04) 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 à sua contribuição para reduzir o custo global da carga.

Funcionamento

Quando um novo comando SQL é submetido ao SGBD, o agente é notificado sobre o novo comando 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.

Se o comando submetido for uma consulta, 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 (Sal04). 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 (Sal04).

Se o comando submetido ao SGBD for uma atualização, executam-se dois procedimentos: uma avaliação idêntica à realizada para consultas e uma avaliação acerca do custo necessária para atualização da estrutura do índice (árvore B^+, por exemplo). Assim, após a aplicação do mesmo procedimento para avaliação de consultas, contabilizam-se os custos de manutenção dos

índices. Estes custos serão debitados do benefício acumulado, tanto de índices hipotéticos quanto de índices reais. 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 (Sal04).

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 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 (Sal04).

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 (Sal04).

Implementação do Protótipo

Em (Sal04) foram implementadas extensões para simulação de índices hipotéticos no SGBD objeto-relacional PostgreSQL versão 7.3. 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.

Resultados Obtidos

A construção da carga de trabalho ocorreu graças ao *toolkit Database Test 2* (DBT-2) provido pela organização *Open Source Development Labs* (OSDL). Este *toolkit* simula uma carga de trabalho baseado no *benchmark* TPC-C (TPC), 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*) (Sal04).

Os resultados obtidos revelaram a eficácia da implementação. Mesmo capturando apenas 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 (Sal04) 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 (Sal04) não se observou nenhuma queda significativa no desempenho do SGBD que tenha sido causada pela presença do Agente de Benefícios.

Comentários Gerais

A abordagem proposta em (Sal04) pode ser classificada como local e intrusiva. A implementação gerada a partir das discussões presentes em (Sal04) 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, gerados pela Heurística de Enumeração de Índices Candidatos (adaptada de (Loh00)), é muito maior que a de índices reais, 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 precisam ser refeitas a fim de portar o agente de benefícios para uma nova versão do PostgreSQL.

3.3.2 Recriação e Remoção Automática de Índices

Em (Morelli06a), os autores estendem os trabalhos propostos em (Sal04, Sal05, Cos05) de 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.

O trabalho apresentado em (Sal04) propõe 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. (Cha04) explica uma heurística capaz de calcular precisamente quanto cada índice contribuiu para redução de custos de uma consulta. Já (Morelli06a) adota uma estratégia intermediária, a qual baseia-se no acompanhamento dos índices criados e na atribuição de um bônus.

Em (Morelli06a), 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.

Ainda que mencionada em (Sal04), a eliminação automática de índices não foi implementada e nem avaliada. Em (Morelli06a) esta funcionalidade foi implementada, porém, realizando uma pequena alteração. Segundo a Heurística de Benefícios (Sal04), 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 (Morelli06a), 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.

Reconstrução Automática de Índices

A partir dos experimentos realizados em (Morelli06a) 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:

- 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 (Morelli06a). Não seria mais adequado reconstruir um índice, ao invés de removê-lo, evitando períodos como 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, (Morelli06a) 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 apenas índices possuindo organização em árvore B^+ (*Btree*).

Uma vez constatada a fragmentação de um índice, causada por sucessivas ocorrências de divisões de páginas (*page splits*), caso decida-se recriá-lo, recomenda-se fazê-lo deixando uma margem para futuras atualizações. Normalmente, os 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 (Morelli06a) 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. Nesta abordagem, quanto mais varreduras um índice fragmentado tiver tido, 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.

A principal desvantagem desta abordagem consiste na possibilidade de um índice permanecer desfragmentado durante um longo período de tempo. Pois, somente quando a Heurística de Benefícios decide remover um determinado índice real é que a Heurística de Reconstrução Automática de Índices é executada.

Comentários Gerais

A abordagem proposta em (Morelli06a) pode ser classificada como local e intrusiva. A implementação gerada a partir das discussões apresentadas restringiu-se ao uso de índices hipotéticos a árvores B^+, limitou-se aos índices secundário e não considerou as restrições quanto ao espaço físico disponível para a criação de índices. Além disso, deve-se frisar que dois novos comandos tiveram de ser criados (*evaluate* e *getsize*), três outros comandos tiveram que ser alterados (*create index*, *reindex index* e *reindex table*) e diversas outras alterações tiveram que ser realizadas junto ao código do PostgreSQL versão 7.3 (uma nova função denominada *pgstatindex*, por exemplo). Todas essas alterações precisam ser refeitas a fim de portar o agente de benefícios e o

agente de *Reindex* para uma nova versão do PostgreSQL. Logo, ao surgir uma nova versão do SGBD PostgreSQL será necessário refazer todas estas alterações (bem como as alterações introduzidas por (Sal04)) 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.

3.3.3 QUIET: Criação Automática de Índices Guiada por Consultas

Em (Sattler03, Kai04) 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 3.2 ilustra a arquitetura do *QUIET*.

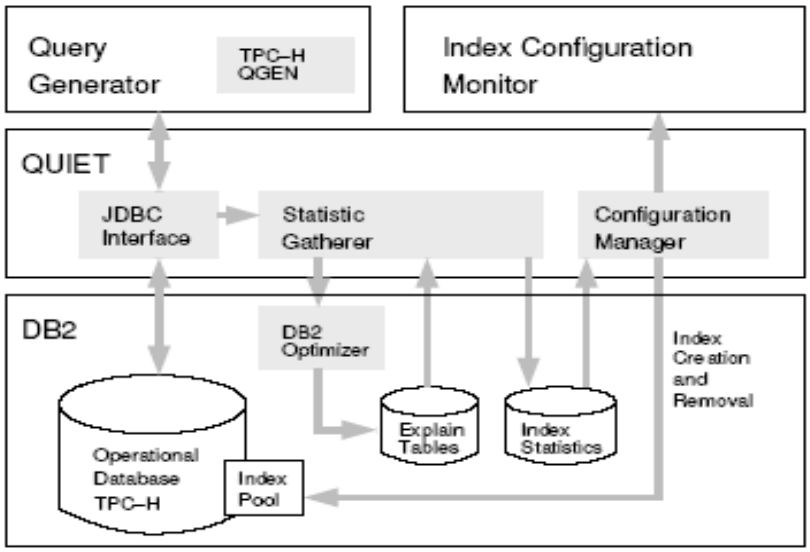

Figura 3.2: Arquitetura da Ferramenta QUIET

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 denomi-

nada "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 potencialmente úteis. Para isso utiliza-se uma estratégia extremamente semelhante à abordagem proposta em (Loh00). 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. 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, 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 algumas aproximações, mas indicam como calcular este benefício particular de forma precisa.

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*. Além disso, 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 e o benefício da configuração atual ultrapassar um determinado valor limite especificado pelo DBA.

Em (Kai04) 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 determinador operador, definido sobre uma determinada tabela, como por exemplo um TABLE SCAN, durante o processamento de uma determinada consulta, para construir um ou 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 registra o novo índice (idx_name) no catálogo do SGBD, como um índice convencional, mas utilizaria um "*flag*" especial 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.

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 financeiramente. 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), restringiu-se a índices baseados em árvores B^+ e considerou somente índices secundários. Outro problema relevante consiste no fato da abordagem não mencionar o acompanhamento dos índices reais, ou seja, uma vez materializado um índice não tem mais o seu benefício atualizado (incrementado).

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

Em (Sch06, Sch07) 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.

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 3.3 mostra a arquitetura da ferramenta *COLT*. A implementação deste protótipo incluiu dois novos módulos no código do PostgreSQL:

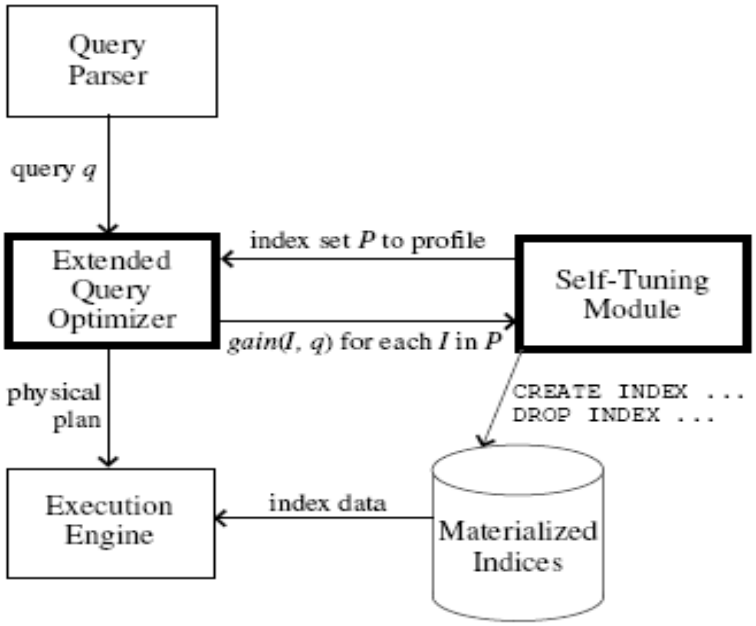

Figura 3.3: Arquitetura da Ferramenta COLT

O otimizador de consultas estendido ("*Extended Query Optimizer*"), denominado EQO, e módulo de auto-sintonia ("*Self-Tuning Module*"), denominado STM.

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, dados uma consulta q e um índice i, o EQO retorna a redução no custo da consulta q caso o índice i fosse utilizado, ou seja, o ganho que o índice i pode proporcionar para 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:

- Se i é um índice materializado, o EQO gera o melhor plano de execução que não utiliza i.
- 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 ($\mathsf{gain}(i, q)$) é calculado da seguinte forma:

$$\mathsf{gain}(i, q) = |C_M - C_i|$$

O Módulo de Auto-Sintonia

O objetivo do módulo de auto-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 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 (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 analisada 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 analisada, 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 subconjunto 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 "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.

Comentários Gerais

A abordagem proposta em (Sch06) pode ser classificada como local e intrusiva. A implementação gerada a partir das discussões apresentadas considerou as restrições quanto ao espaço físico disponível para a criação de índices e a estabilidade da configuração de índices selecionada. Contudo, uma mesma consulta pode precisa ser otimizada uma vez para cada índice candidato, o que pode gerar grande sobrecarga. Além disso, (Sch06) restringiu-se ao uso de índices hipotéticos a árvores B^+, limitou-se aos índices secundário e não tratou o problema da fragmentação das estruturas de índices. O protótipo foi implementado junto ao código do PostgreSQL. Os autores não comentam sobre a quantidade de modificações realizadas junto ao código fonte do PostgreSQL. Todavia, estas alterações precisarão ser refeitas a fim de portar o protótipo para uma nova versão do PostgreSQL.

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

Em (Luh07) 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 continuamente e coleta estatísticas acerca dos índices reais e hipotéticos, e, com base nas informações coletadas, soluciona periodicamente o problema da seleção de índices (*Index Selection Problem - ISP*[1]).

O aspecto dinâmico da sintonia automática de índices é descrito em (Luh07) de acordo com o modelo genérico de sintonia automática, o qual consiste em um ciclo de Observação, Predição, e Reação, como inicialmente proposto em (Weikum94). 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* ("*on-the-fly*")

[1] O ISP foi formalmente definido na Seção 5.2.1

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*. 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.

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 (Loh00). Em seguida, estima-se o tamanho de cada índice candidato 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.

O benefício do conjunto de índices candidatos, para uma determinada consulta Q, pode ser calculado com base no custo de execução de Q nas duas otimizações anteriores. Logicamente, os índices que pertencem ao conjunto de índices candidatos podem contribuir de forma diferente para o benefício total. Para atribuir a cada índice candidato parte do benefício total, os autores utilizam aproximação (baseada nas estratégias apresentadas em (Kai04)).

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.

Em ((Luh07)), os autores observam ainda que 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 $R(A, B)$.

Para lidar com o aspecto temporal da abordagem proposta, os autores utilizam o conceito de época (Kai04), 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

há bastante tempo 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 (Kai04, Sal04, Sal05, Cos05, Morelli06a, Lif06)) pode gerar uma sobrecarga desnecessária e resultar em configurações instáveis. Nesse 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 (Luh07), 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.

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)[2] 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

[2]O ISP foi formalmente definido na Seção 5.2.1

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. Este algoritmo utiliza como entrada o conjunto de todos os índices (materializados ou hipotéticos) ordenados pelo benefício relativo (benefício do índice dividido pelo seu tamanho). A saída do algoritmo é uma nova configuração de índices $\bar{C}$.

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 pode 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 Fase de Reação: Criação de Iindices *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, por exemplo, 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 (Graefe00). 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.

Comentários Gerais

A abordagem proposta em (Luh07) 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 (Luh07) 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 (Luh07, Idreos07a, Idreos07b).

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

Em (Bruno07a, Bruno07b) 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 3.4 ilustra os componentes do SQL Server que foram modificados ou adicionados a fim de possibilitar a sintonia automática do projeto físico. O gerenciador de meta-dados 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 internas 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 desviada para o módulo de análise de índices (*Index analysis*) a fim de identificar um conjunto de índices candidatos relevantes 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 Adjustment*" (Figura 3.4). 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.

Comentários Gerais

A abordagem proposta em (Bruno07a, Bruno07b) foi implementada junto ao código fonte do SQL Server 2005, podendo ser classificada como

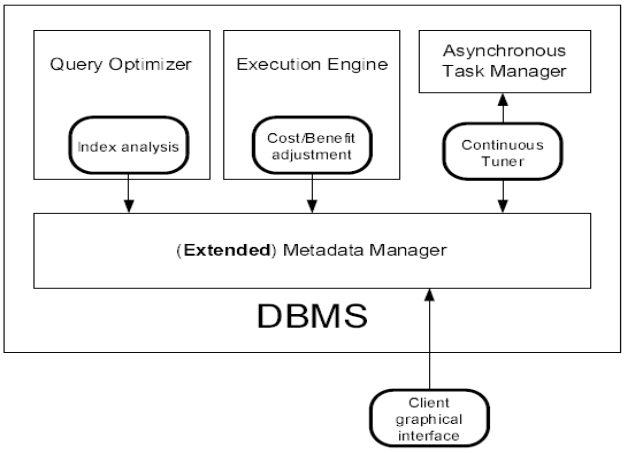

Figura 3.4: Arquitetura da Ferramenta *AutoAdmin Online*

local e intrusiva. Esta abordagem considera as restrições quanto ao espaço físico disponível para a criação de índices, a estabilidade da configuração de índices selecionada, a atribuição proporcional de benefícios, utiliza uma única heurística, cada consulta é otimizada uma única vez, além de gerar uma pequena quantidade de índices candidatos. Contudo, considerou apenas a utilização de índices baseados em árvores B^+ e secundários. Além disso, não tratou o problema da fragmentação das estruturas de índices.

3.3.7 Análise Comparativa

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 a manutenção de índices secundários e organizados exclusivamente em árvore B^+, não levam em consideração os conceitos de generalidade e relevância, 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*"), conforme mostraremos a seguir. Uma análise completa do estado da arte em auto-sintonia do projeto físico de banco de dados pode ser encontrada em (Monteiro07b).

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). Indica os tipos de estruturas de acesso utilizadas pela abordagem (índice primário, índice secundário, visões materializadas, partições de tabelas, etc) e a organização física destas estruturas (árvores B^+, índice *hash*, etc);
- Ações de gerenciamento executadas sobre as estruturas de acesso utilizadas (C2). Aponta que ações de gerenciamento são suportadas pela solução estudada, que podem ser: criação, remoção e reorganização de estruturas de acesso;
- Nível de acoplamento da solução ao SGBD utilizado (C3). Indica o grau de acoplamento (integração) do código da abordagem analisada com o código do SGBD utilizado;
- SGBDs suportados (C4). Descreve que SGBDs podem ser utilizados pela abordagem estudada;
- Quantidade de heurísticas utilizadas (C5). Indica a quantidade de heurísticas distintas utilizadas pela solução analisada;
- Quantidade de índices hipotéticos gerados (C6). Aponta a quantidade de índices hipotéticos gerados pela solução. Uma grande quantidade de índices hipotéticos pode gerar uma sobrecarga (*overhead*) no gerenciamento destes índices;
- Modelo de custos utilizado (C7). Indica se o modelo de custos utilizado pelas heurísticas é interno (ou seja, é o próprio modelo de custos utilizado pelo SGBD) ou externo (um modelo de custos independente de SGBD);
- Consideração de restrições de armazenamento (C8). Avalia se a abordagem analisada leva em consideração as restrições de armazenamento;
- Atribuição proporcional de benefícios (C9). Indica se a abordagem atribui benefício de forma proporcional, se não o faz, ou se utiliza uma solução aproximada;
- Número de vezes que cada tarefa (consulta) é otimizada (C10). Aponta a quantidade de vezes que uma mesma tarefa é otimizada durante o cálculo dos benefícios;
- Quantidade de comandos criados ou alterados junto ao código fonte do SGBD utilizado (C11). Representa uma medida da intensidade de alterações necessárias junto ao código fonte do SGBU utilizado. Quanto maior este valor, maior será o acoplamento da abordagem analisada com o código fonte do SGBD;
- *Benchmark* utilizado na avaliação da solução (C12). Descreve quais os *Benchmark* que foram utilizados para a valiação de desempenho da solução;

– Consideração da estabilidade da configuração de índices (C13). Indica se a abordagem analisada fornece alternativas para reduzir a volatilidade das configurações selecionadas;

– Utilização do conceito de generalidade (C14). Indica se o conceito de generalidade foi ou não utilizado;

– Utilização do conceito de relevância (C15). Indica se o conceito de relevância foi ou não utilizado;

– Avaliação da pré-criação de índices envolvidos em chaves primárias e estrangeiras (C16). Indica se abordagem estudada analisou a alternativa de criar previamente as estruturas de índices envolvidas em chaves primárias e estrangeiras;

A Tabela 3.1 apresenta, de forma sumarizada, uma comparação entre as soluções existes. Esta comparação baseia-se nas características previamente listadas [3].

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	Postgre	Postgre	DB2	Postgre	Postgre	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	Aprox.	Aprox.	Aprox.	Aprox.	Sim
C10	3	3	2	várias	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 3.1: Análise Comparativa das Abordagens para Manutenção Automática de Índices.

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

3.3.8 Resumo do Capítulo

Neste capítulo, foram discutidas as principais abordagens, encontradas na literatura, para o projeto físico automático de bancos de dados. Uma análise comparativa detalhada destas abordagens também foi apresentada. Além disso, justificamos e propomos uma nova classificação para as pesquisas em auto-sintonia de bancos de dados. No próximo capítulo será apresentada uma abordagem não-intrusiva para a manutenção automática do projeto físico de bancos de dados relacionais.