2 Avaliação da Eficácia da Heurística de Benefícios com Cargas OLAP

O presente capítulo oferece um apanhado geral sobre o estado da arte da área de estudos denominada auto-sintonia, consistindo em propostas que reduzam as intervenções humanas em trabalhos de ajustes no desempenho (seção 2.1). Discute-se auto-sintonia global, estudada em [26] (seção 2.2), bem como a auto-sintonia local, discutida em [32] (seção 2.3), que serviu como ponto de partida para as idéias discutidas neste trabalho de pesquisa.

A partir da quarta seção, discutem-se temas únicos levantados por esta Dissertação. Inicia-se com a estruturação de uma bateria de testes inspirada no *benchmark* TPCH, que servirá para avaliar a eficácia da ferramenta de criação automática de índices gerada a partir das idéias propostas em [32]. Surgirão questões a partir de obstáculos enfrentados. Particularmente, discute-se a influência da seletividade de consultas nas decisões tomadas pelo otimizador do SGBD. Também apresenta-se um novo comando desenvolvido para o SGBD de código aberto PostgreSQL, cuja utilização será fundamental para realização da bateria de testes.

O restante do capítulo apresenta resultados obtidos para a bateria de testes estruturada na seção 2.4 em três SGBDs: PostgreSQL, SQL Server e Oracle.

Na seção 2.8 encerra-se o presente capítulo tecendo comentários gerais sobre as discussões realizadas.

2.1 Trabalhos Correlatos

Um panorama geral sobre o estado das pesquisas em auto-sintonia até 2004 é apresentado em [23]. Após detalhar os princípios nos quais baseia-se a auto-sintonia, propõe-se uma classificação das linhas de pesquisa em dois grandes grupos: auto-sintonia global e auto-sintonia local. O primeiro englobaria projetos preconizando a construção de um SGBD levando em conta sintonia automática desde seu projeto, bem como aqueles propondo alterações em SGBDs

PUC-Rio - Certificação Digital Nº 0410827/CA

prontos. O segundo grupo consistiria em iniciativas de automação para o projeto físico (criação de objetos), alocação de dados, controle de cargas, substituição de páginas, ajustes de buffers (memória) e refino de estatísticas.

Há vários anos o projeto AutoAdmin [1] da Microsoft vem publicando artigos a respeito da sintonia automática. Em [8] apresenta-se a ferramenta Index Tuning Wizard para Microsoft SQL Server 7.0, onde, a partir de uma análise de uma carga de trabalho (*workload*), sugere-se a criação de índices. Já em [8] detalha-se a arquitetura interna para obtenção de índices hipotéticos utilizados na geração automática de estatísticas sob o mesmo SGBDR. Dois anos depois, os trabalhos evoluíram para a detecção automática, não apenas de índices, mas também de visões materializadas [2]. Acompanhando o lançamento da versão 2005 do SQL Server, o processo foi estendido para que também sejam sugeridas particionamento em grandes tabelas [3]. Uma proposta que simplifica e uniformiza o conjunto de técnicas atuais voltadas para automação do projeto físico de Bancos de Dados constitui a maior contribuição de [4]. Finalmente, [18] propõem uma forma alternativa de análise de cargas de trabalho, baseado em amostragens.

Em [7] formaliza-se a seleção de índices como um problema de otimização e demonstra-se que, tanto a escolha de uma configuração ótima de índices non-clustered como clustered, são problemas NP-difícil. Também sugere-se uma heurística capaz de atribuir benefícios individuais a cada índice participando 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 à soma dos benefícios dos índices sobre os campos constituintes.

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. Com características semelhantes aos temas tratados nesta proposta, destacam-se [24], onde sugere-se a heurística de escolha de índices candidatos implementada em [32] e [44], detalhando a criação automática de visões materializadas. Exemplos de auto-sintonia com DB2 são listados em [13]; já [34] apresenta recursos de configuração automática de memória para DB2. [16]e [25] exploram, respectivamente, a coleta automática de estatísticas e a utilização de visões materializadas para balanceamento de cargas.

PUC-Rio - Certificação Digital Nº 0410827/CA

A Oracle tem investido bastante nos últimos anos na crescente automação de seu SGBD. A última versão, 10g RII, oferece vários recursos que permitem a criação de *scripts* contendo instruções que dinamicamente ajustem o ambiente, segundo os níveis de utilização [5]. Também [11] discute auto-sintonia descrevendo o funcionamento da ferramenta ADDM (*Automatic Database Diagnostic Monitor*). Ela permite realizar o acompanhamento e ajuste de desempenho de forma automática em bases Oracle.

Em [35] 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. Entretanto o foco reside em bases OLAP, o que desconsidera a manutenção de índices.

Shasha demonstra, através de testes práticos, que a não manutenção periódica de índices faz com que o desempenho seja degradado com o tempo[33].

O SGBD de código aberto PostgreSQL oferece poucos recursos de auto-sintonia, com destaque para a ferramenta *pg_autovacuum*. Ela, periodicamente, compacta tabelas eliminando fisicamente tuplas que tenham sido marcadas para exclusão. [32] e [26] propõem a utilização de auto-sintonia e geraram protótipos utilizando PostgreSQL.

2.2 Auto-sintonia Global de SGBDs

A pesquisa referente à dissertação de mestrado de Anolan Milanés [26] e [27] propõe duas arquiteturas, baseadas em agentes, onde todos os parâmetros de sintonia de um SGBD seriam ajustados automaticamente, via agentes de software. Parte-se do princípio que trabalhar a sintonia de um SGBD focando apenas 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.

PUC-Rio - Certificação Digital Nº 0410827/CA

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

- 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;

- 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

- 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.

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

Duas arquiteturas foram propostas:

- Centralizada: há um agente, dito 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 [19], este agente teria apenas as camadas de Crença, Raciocínio e Colaboração.

- 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.

Escolheu-se a opção Centralizada para realização de testes e obtenção de resultados. A implementação foi conduzida de forma unificada ao núcleo do SGBD.

PUC-Rio - Certificação Digital Nº 0410827/CA

2.3 Criação Autonômica de Índices em Bancos de Dados

A pesquisa referente à dissertação de mestrado de Marcos Salles [32], classificada como auto-sintonia local [23], propõe uma estratégia para criar índices automaticamente baseada em duas heurísticas.

A primeira, denominada Heurística de Escolha de Índices Candidatos, ocupa-se em analisar comandos fornecidos pelo SGBD, e enumerar índices que potencialmente melhorariam o comando. Aqueles índices candidatos que se provarem úteis passam a existir como hipotéticos. Este processo de seleção de índices candidatos inspirou-se numa heurística proposta em [24]. Como esta presume a existência de índices do tipo "what-if", tais como propostos em [9], houve a necessidade de estender a funcionalidade do SGBD PostgreSQL para que o otimizador levasse em conta a presença de índices hipotéticos [22]. 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.

A segunda, denominada Heurística de Benefícios, 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 a carga de benefícios seja tão grande a ponto de compensar o custo da criação do índice, este deixa de ser hipotético e transforma-se em índice real. No próximo ítem apresentam-se mais detalhes desta heurística.

Heurística de Benefícios

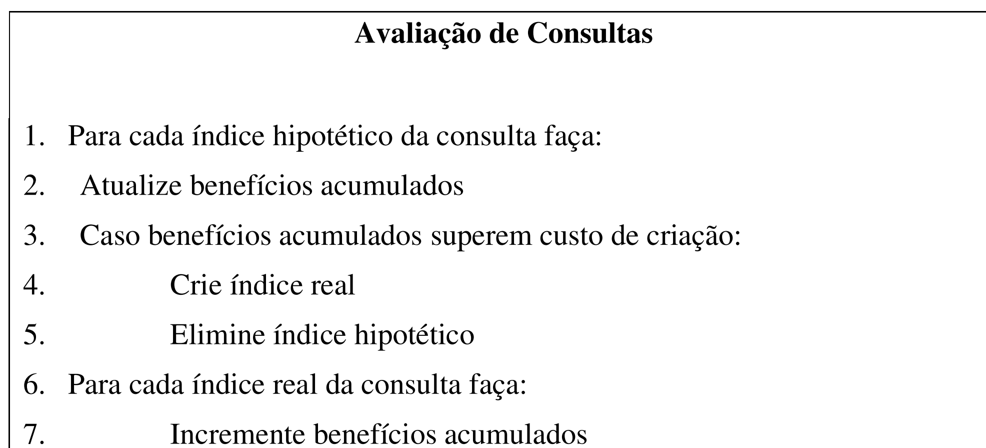
A Heurística de Benefícios foi inicialmente proposta em [6], porém nunca tinha sido testada para uso efetivo em um SGBD. Aconteceram diversas extensões em [32], tais como a apresentação de equações para cálculos dos benefícios e refinamento das estratégias de avaliação de consultas, que tornaram a heurística mais realista.

Com base no plano de execução gerado pelo SGBD (enriquecido por índices hipotéticos), atualizam-se os benefícios dos índices armazenados. Para todos os índices hipotéticos, cujos benefícios acumulados superem os custos de

PUC-Rio - Certificação Digital Nº 0410827/CA

criação, disparam-se ações para transformá-los em índices reais. Analogamente, a carga de benefícios de um índice hipotético será reduzida, caso este potencialmente aumente o custo de um comando. Tipicamente, tratam-se de atualizações onde a presença de índices representaria um custo adicional, já que há necessidade em atualizá-los.

A Figura 2.1 revela a estratégia adotada pela Heurística de Benefícios para consultas, cujo custo tenha diminuído graças à presença de índices.



Avaliação de Consultas

1. Para cada índice hipotético da consulta faça:
2. Atualize benefícios acumulados
3. Caso benefícios acumulados superem custo de criação:
4. Crie índice real
5. Elimine índice hipotético
6. Para cada índice real da consulta faça:
7. Incremente benefícios acumulados

Figura 2.1: estratégia de Avaliação de consultas.

Observações:

- A atualização dos benefícios para cada índice hipotético (linha 2) ocorre ao comparar a diferença entre o custo do comando capturado e o novo custo obtido com a presença de índices hipotéticos. Caso esta diferença exista, todo índice que tenha participado ganha um incremento em sua carga de benefícios igual a esta diferença.
- O mesmo mecanismo aplica-se no incremento de benefícios de índices criados (linha 7). Caso estes participem de comandos e suas contribuições causarem queda no custo do comando, cada índice real recebe um incremento em seus benefícios.

A Figura 2.2 revela a estratégia adotada pela Heurística de Benefícios para atualizações, cujo custo tenha aumentado graças à presença de índices.

Avaliação de Atualizações

PUC-Rio - Certificação Digital Nº 0410827/CA

1. Para cada índice afetado pela atualização faça:
2. Decremente benefícios acumulados
3. Caso índice seja real:
4. Caso índice deva ser removido:
5. Remova índice criado

Figura 2.2: estratégia de Avaliação de atualizações

Observação:

- A verificação quanto a eliminação de um índice criado (linha 4) testa se os benefícios acumulados são inferiores ao custo de eliminação;

Arquitetura da Implementação

A decisão quanto à utilização do paradigma de desenvolvimento do código recaiu sobre o emprego de agentes de software, entretanto, deve-se ressaltar que a escolha poderia ter sido pelo emprego de objetos, por exemplo. Inclusive, não existiu uma preocupação em obedecer rigidamente às características de agentes, tais como apresentadas em [42]. A implementação da ferramenta de criação automática de índices ocorreu utilizando-se o *framework* baseado em agentes proposto em [19] (Figura 2.3).

[32] propõe duas arquiteturas: uma com um único agente (**mono-agente**) e outra com vários (**multi-agentes**). Na primeira, o único agente, denominado Agente de Benefícios, apresenta apenas as camadas: *Sensor*, *Crença*, *Raciocínio* e *Ação*. As interações com o SGBD acontecem na captura de comandos (camada *Sensor*), ao requisitar um plano de execução para um comando enriquecido por índices candidatos e ao disparar uma criação ou destruição de índices (camada *Ação*).

Na segunda arquitetura há um sistema de três agentes: o primeiro coleta comandos; o segundo simula configurações hipotéticas e o terceiro cria e destrói índices reais. Esta arquitetura foi apenas discutida, não ocorrendo sua implementação.

PUC-Rio - Certificação Digital Nº 0410827/CA

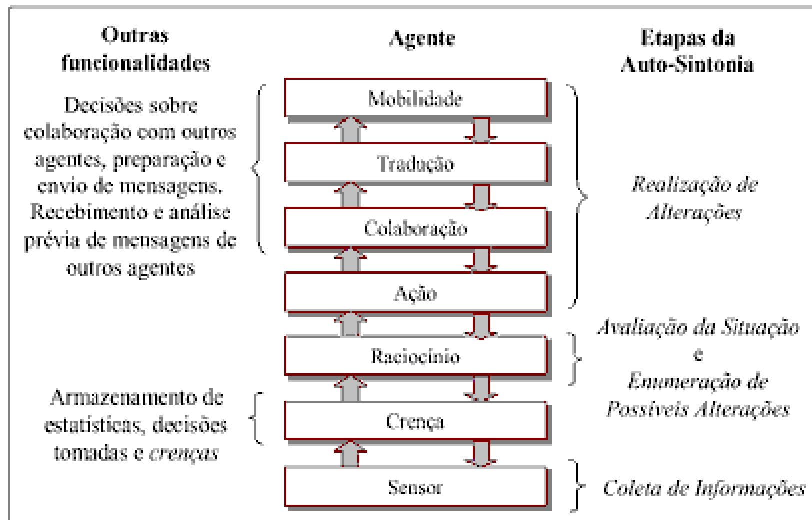


Figura 2.3: framework de construção de agentes de [KPP+99].

Como a implementação e a coleta de resultados aconteceu na arquitetura **mono-agente**, segue uma breve descrição dos papéis desempenhados em cada camada utilizada pelo Agente de Benefícios.

Sensor: checa permanentemente a chegada de um lote, no qual encontra-se uma estrutura de dados com tabelas e índices, além do texto do comando SQL corrente. Uma vez acusada a recepção, encaminha-se o conjunto à camada *Crença*.

Crença: divide-se em dois componentes: o primeiro acusa o recebimento proveniente da camada *Sensor* e notifica a camada *Raciocínio* que um evento ocorreu. O segundo possui estruturas onde armazenam-se os benefícios acumulados referentes aos índices hipotéticos criados. Estes dados históricos comportam conjuntos de índices candidatos, seus benefícios acumulados e seus custos de criação.

Raciocínio: local onde concentra-se a inteligência do agente, já que responsabiliza-se pela análise dos dados da camada de crenças e tomada de decisões que podem transformar-se em instruções à camada *Ação*. Também divide-se em dois componentes: o primeiro implementa a Heurística de Enumeração de Índices Candidatos. Uma vez recebida a notificação proveniente da camada *Crença* que um evento ocorreu, analisa-se o comando corrente, elabora-se a lista de índices candidatos e prepara-se uma requisição ao Otimizador do SGBD para que um novo plano de execução seja elaborado, onde estejam presentes os índices candidatos.

PUC-Rio - Certificação Digital Nº 0410827/CA

O segundo componente implementa a Heurística de Benefícios, apresentada no ítem anterior.

Ação: realiza cinco procedimentos possíveis, a partir de requisições da camada *Raciocínio*:

- Criação de índices hipotéticos;
- Criação de índices reais;
- Eliminação de índices hipotéticos;
- Eliminação de índices reais;
- Otimização de um comando enriquecido por índices hipotéticos.

Na verdade, as ações concretizam-se no SGBD. O Agente de Benefícios simplesmente requisita que estas aconteçam. Daí a necessidade de implementação com alteração do código fonte do SGBD.

PUC-Rio - Certificação Digital Nº 0410827/CA

Resultados Obtidos

A construção da carga de trabalho ocorreu graças ao *toolkit Database Test 2* (DBT-2) provido pela organização *Open Source Development Labs* [29]. Este *toolkit* disponibiliza *scripts* que permitem a criação de uma carga de trabalho baseada no *benchmark* TPC-C [TPCC], que favorece a incidência de curtas transações. Há consultas e alterações em pequenas quantidades de tuplas, simulando um ambiente com características OLTP (*On Line Transaction Processing).*

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 aqueles mais representativos, já que todos os índices importantes foram criados. Inclusive, aconteceu a construção de um índice ausente no conjunto inicialmente proposto pelo *toolkit*, que melhorou o desempenho de consultas nas quais participou.

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. Não se observou, contudo, nenhuma queda significativa no desempenho do SGBD que tenha sido causada pela presença do Agente de Benefícios.

A implementação gerada a partir das discussões presentes em [32] para o Agente de Benefícios, não contemplou a eliminação de índices reais. Mas, como registrou-se uma participação maior de consultas em relação a atualizações, esta carência não se mostrou relevante.

A partir da próxima seção, desvincula-se a discussão dos assuntos apresentados em [32].

2.4 Agente de Benefícios e TPC-H

A carga de trabalho utilizada em [32], gerada com auxílio do *toolkit* DBT-2, possui características de OLTP. Além disto, trata-se de um conjunto estável de comandos, onde sempre acontecem as mesmas instruções. Este fato levantou questionamentos sobre a eficácia do Agente de Benefícios caso fosse submetido a uma carga de trabalho com características *ad-hoc*, isto é, onde os comandos submetidos tivessem variáveis, cujos valores mudariam a cada execução.

Com base nesses questionamentos, resolveu-se submeter o Agente de Benefícios a uma carga de trabalho alternativa, do tipo OLAP, com características *ad-hoc*. Objetivou-se verificar se a implementação de [32], quando submetida a uma carga OLAP, seria tão eficaz quanto a submissão com carga OLTP.

A construção da carga de trabalho alternativa ocorreu graças ao *toolkit Database Test 3* (DBT-3), também provido pela organização *Open Source Development Labs* [29]. Este *toolkit* simula uma carga de trabalho baseado no *benchmark* TPC-H [39], que favorece a incidência de consultas onerosas, simulando um ambiente com características OLAP (*On Line Analytical Processing).*

A utilização do *toolkit* DBT3 proporcionou a criação da base de dados segundo as diretrizes do TPC-H, com *scripts* das consultas adaptadas à sintaxe PostgreSQL, bem como viabilizou a geração de grupos de consultas a serem executadas.

O Apêndice A desta dissertação explica em detalhes a constituição do *benchmark* TPC-H (8 tabelas, 23 índices, 22 consultas), bem como as adaptações necessárias para obtenção de resultados que comprovassem a eficácia da Heurística de Benefícios, quando submetida a uma carga de trabalho alternativa.

PUC-Rio - Certificação Digital Nº 0410827/CA

A obtenção de resultados para o SGBD PostgreSQL pôde ser concluída após realizar duas alterações importantes. Houve a necessidade em criar um novo comando e ajustes tiveram que acontecer nas consultas TPC-H, motivados pelo problema da seletividade. Os detalhes aparecem nas duas próximas subseções.

EVALUATE

O *toolkit* DBT-3 sugere a presença de 23 índices, dos quais apenas cinco não estão envolvidos com chaves primárias ou estrangeiras. Desta forma, preferiu-se executar o Agente de Benefícios em um ambiente totalmente desprovido de índices para observar quantos de cada tipo seriam criados (integridade e desempenho). Executou-se cada consulta de forma independente, ou seja, antes de cada execução reiniciava-se o PostgreSQL e eliminavam-se quaisquer índices que porventura existissem.

Em vários casos, foi constatada a competência do Agente na criação dos índices necessários, entretanto, a duração das consultas permanecia tão alta quanto as execuções que não utilizassem índices. Apesar da correta identificação por parte do Agente dos índices candidatos, e estes apresentarem benefícios muito superiores aos custos de suas criações, o otimizador não esperava tempo suficiente para que o Agente informasse tal necessidade. Desta forma, o otimizador disparava os comandos sem a presença dos índices sugeridos. Assim, interrompendo-se a execução da consulta e criando-se manualmente os índices sugeridos pelo Agente, ao submetê-la novamente, o tempo de resposta era o esperado. Deve-se ressaltar que trata-se de uma situação bastante realista, já que, no cotidiano, seria exatamente assim o trabalho de um DBA experiente: ao constatar a presença de alguma consulta onerosa, esta seria interrompida para análise de seu plano de execução e criação de índices.

Nesse contexto de criação totalmente automática de índices, concluiu-se a necessidade de construir uma ferramenta de apoio, batizada como **evaluate**, que repassaria uma dada consulta ao agente, mas eliminando-se a fase de execução. Desta forma, seria possível avaliar todas as consultas, sem as etapas mais custosas.

Por exemplo, suponha o comando ilustrado na Figura 2.4, que referencia a maior tabela do Esquema TPC-H:

PUC-Rio - Certificação Digital Nº 0410827/CA

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

Figura 2.4: exemplo de execução de **evaluate**, cujo argumento (**select**) tem sua execução inibida.

Supondo que a tabela **lineitem** fosse muito grande e não tivesse nenhum índice, a execução do comando **select** provocaria uma varredura completa na tabela (*full table scan*), o que seria muito mais demorado caso houvesse um índice sobre os campos **orderkey** e **linenumber**. Entretanto, graças à nova palavra reservada **evaluate**, inibe-se a última etapa de processamento da consulta, a saber, a execução. Em resumo, o comando acima pode ser dividido nas partes a seguir:

i. Análise Sintática (*parsing*): verificam-se as palavras reservadas utilizadas. Nesse caso, há cinco: **evaluate**, **select**, **from**, **where** e **and**. O resultado desta etapa denomina-se **parse tree**;
ii. Análise Semântica (*transformation*): checada a permissão de acesso aos objetos envolvidos: cinco campos da tabela **lineitem.** O resultado desta etapa denomina-se **query tree**;
iii. Reescrita (*rewriting*): otimizador eventualmente altera a *query tree* em busca de pequenas otimizações (mudança na ordem das tabelas enumeradas na cláusula **from**, por exemplo);
iv. Planejamento (*planning*): cria-se um plano de execução onde constem estruturas auxiliares (índices) que acelerem a execução.

Além de criar uma nova instrução, **evaluate**, impediu-se a execução do argumento devido à implantação de um desvio no código fonte do PostgreSQL. Desta forma, ao invés de executar o argumento, este é apenas encaminhado ao Agente de Benefícios.

Uma aplicação interessante e adicional do novo comando seria a possibilidade de analisar uma carga de trabalho a posteriori, tal qual preconizam [8] e [24]. Ao invés de submeter os comandos ao Agente de Benefícios à medida

PUC-Rio - Certificação Digital Nº 0410827/CA

que são executados, seria criada uma carga contendo vários comandos não executados. Estes poderiam receber a nova palavra reservada **evaluate** em seus inícios para que depois houvesse nova submissão da carga. O resultado deste processo seria um conjunto de índices criados pelo agente que atenderia aos comandos em questão.

Dessa forma, o novo comando procura desempenhar o papel dos assistentes de apoio ao desempenho (*tuning wizards*) em ferramentas comerciais tais como Oracle 10g [30] e SQL Server 2005 [36]: quando percebe-se uma consulta demorada, provoca-se sua interrupção e utiliza-se o assistente com intuito de conhecer sua sugestão no que concerne à criação de índices. Uma vantagem de **evaluate** em relação aos *tuning wizards*, entretanto, seria sua praticidade, já que não seria necessário realizar nenhuma configuração prévia.

Um resultado equivalente ao obtido com o novo comando **evaluate**, poderia ter sido obtido através da simples utilização do parâmetro PostgreSQL **statement_timeout**, que interrompe a execução de quaisquer comandos após um intervalo medido em milisegundos. Comandos de longa duração teriam suas execuções interrompidas. Porém, o agente prosseguiria com seu trabalho criando os índices necessários. Na próxima execução, a duração não seria tão longa, já que os índices já estariam presentes. O inconveniente desta alternativa seria a dificuldade em estipular o intervalo mínimo suficiente, após o qual os comandos seriam interrompidos, daí a opção pelo novo comando.

Seletividade

Quanto menos tuplas retornadas por uma consulta sobre uma tabela, maior será a seletividade desta consulta. Otimizadores de SGBDs levam este fato em conta ao avaliarem utilizações de índices. Por exemplo, imagine uma tabela T possuindo um milhão de tuplas, um determinado campo denominado UF e um índice sobre este atributo. Supondo que 999.999 tuplas possuam UF = ‘RJ’, o índice somente seria utilizado quando fosse solicitada a única tupla com valor diferente de ‘RJ’.

Quando o DBA depara-se com uma consulta custosa, além de avaliar a ausência de índices, também deve-se preocupar com a seletividade, para que não sejam criados índices que não venham a ser aproveitados. Observando-se o

PUC-Rio - Certificação Digital Nº 0410827/CA

comportamento do Agente de Benefícios quando submetido a uma carga de trabalho TPC-H, perceberam-se três fatos:

i. Em consultas com baixa seletividade, não se criavam índices; nem mesmo hipotéticos;
ii. Em consultas com alta seletividade, índices eram criados na primeira execução;
iii. Em consultas com seletividade média, índices hipotéticos eram criados e, após algumas execuções, aconteciam as transformações de hipotéticos para reais.

Trata-se, portanto, de uma forma de atuação similar a um DBA; consultas com seletividade média devem ser testadas várias vezes para que se convença da necessidade de criar um índice.

Uma diferença marcante entre as cargas de trabalho geradas a partir dos *benchmarks* TPC-C e TPC-H, reside no fator previsibilidade. Se por um lado os comandos TPC-C não sofrem variações em seus filtros (cláusula *where* dos comandos *select, delete* e *update*), nem na ordem de execução dos comandos, isto não ocorre com instruções TPC-H. Este fato faz com que a seletividade das consultas varie entre execuções, levando o otimizador a nem sempre utilizar índices para resolvê-las.

Ao submeter as consultas TPC-H ao Agente de Benefícios, percebeu-se que a quantidade de execuções necessárias para que um índice fosse criado variava bastante. Explica-se este problema pela atuação do Agente de Benefícios, que atribui mais, ou menos, benefícios a um índice hipotético, dependendo da seletividade apresentada pela consulta, na qual houve participação deste índice. A Figura 2.5 revela o comportamento observado ao variar o filtro de uma consulta a uma grande tabela.

PUC-Rio - Certificação Digital Nº 0410827/CA

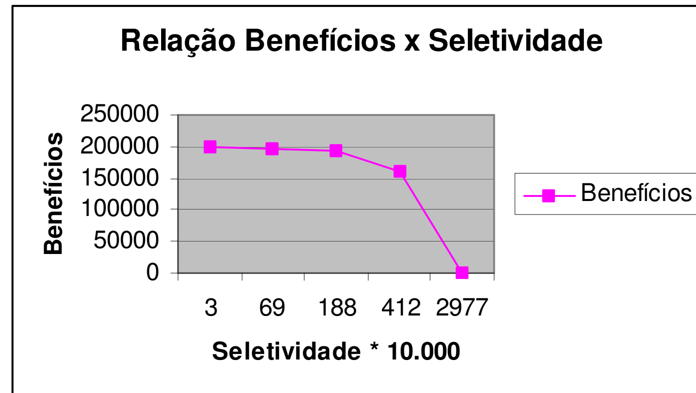


Figura 2.5: o otimizador proporciona ganhos decrescentes à medida que piora a seletividade

A Figura 2.5 revela que há um grande benefício quando a seletividade vale 0,0003%; e, a partir de 0,2977% a utilização de índices não proporciona ganho algum. Isto explica porque otimizadores preferem realizar varreduras completas em tabelas (*full scans*) a utilizar índices que proporcionem baixa seletividade.

Dada a variação entre execuções de consultas TPC-H, resolveu-se escolher um conjunto composto por seis consultas consideradas mais representativas e ajustá-las para que a atuação do Agente de Benefícios fique evidenciada ao máximo. A relação das consultas escolhidas aparece no Apêndice A.

PUC-Rio - Certificação Digital Nº 0410827/CA

Trabalhos Relacionados com Avaliações TPC-H

Antes de elaborar a bateria de testes, buscou-se em trabalhos relacionados como se utiliza o *benchmark* TPC-H na prática e no contexto de sintonia automática de índices.

[9] discute técnicas eficientes para implementação de índices hipotéticos. Utilizou-se TPC-D (*benchmark* precursor de TPC-H) com uma base de 1 GB e, para as cinco consultas mais custosas, de um total de dezessete, realizou-se uma única execução do grupo de consultas. Ao contrário de [32], onde os testes ocorrem determinando-se uma duração fixa (30, 60 e 90 minutos), o fator "tempo" não foi utilizado como variável de estudo, ou seja, não pretendeu-se medir níveis de vazão (consultas executadas por minuto). Simplesmente executaram-se as consultas escolhidas e observaram-se resultados.

[2] apresenta uma ferramenta que cria automaticamente, não apenas índices, como também visões materializadas. Realiza seus experimentos com duas bases, uma extraída de TPC-H com 1 GB e outra interna à Microsoft. Existem seis cargas de trabalho (*workloads*), porém as consultas advém de outros *benchmarks*, tais como TPCH-UPD25 e TPCH-UPD75, que apresentam maiores incidências de atualizações (25% e 75%, respectivamente).

[24] introduz as facilidades oferecidas pelo DB2 em criar seus próprios índices em uma base TPC-D com 1 GB. Partindo-se de um conjunto de dezessete consultas submetidas, a ferramenta proposta, **db2advis**, foi capaz de otimizar quatorze.

[7] explica uma heurística capaz de calcular precisamente quanto cada índice contribuiu para redução de custos de uma consulta. Executam-se as vinte e duas consultas do *benchmark* TPC-H, em um conjunto de tabelas possuindo apenas índices de integridade (chaves primárias). Vale destacar a restrição à área destinada a novos índices, limitada em 3 GB. Esta preocupação não constou em [32].

[3]mostra como a ferramenta DTA (*Database Tuning Advisor)* sugere a criação automática de índices, visões materializadas e partições no SQL Server 2005. Parte-se de uma base extraída do *benchmark* TPC-H com 10 GB e já com dezoito índices criados (todas as chaves primárias e estrangeiras). Outra experiência usa uma base menor, com 1 GB, onde executam-se as vinte e duas consultas seqüencialmente apenas uma vez.

[14] avalia os benefícios acarretados pela utilização de algoritmos *Hash-Join* e *Merge-Join* pelo otimizador, que somente utilizava *Nested Loop*. Em bases TPC-H com 100 MB e 1 GB, executa as vinte e duas consultas de forma seqüencial.

[35] propõe 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. Há duas bases de dados TPC-H: uma com 100 MB e outra com 1 GB. As duas cargas de trabalho executam as vinte e duas consultas quatro vezes em um experimento, e cem consultas aleatórias, em outro teste.

Analisando os trabalhos relacionados, conclui-se o seguinte:

PUC-Rio - Certificação Digital Nº 0410827/CA

- Não há menção explícita aos testes Power ou Throughput, por exemplo (veja Apêndice A). Simplesmente aproveita-se a base de dados e as consultas de algum *benchmark*.
- Nada indica que houve escolhas aleatórias para ordem de execuções das consultas, bem como para valores utilizados em filtros.

Essas observações avalizam o mecanismo escolhido nesta dissertação para testar o Agente de Benefícios, onde escolheram-se seis consultas e realizaram-se ajustes para que a seletividade fosse a melhor possível.

2.5 Resultados Experimentais com PostgreSQL

A bateria de testes utilizando o SGBD PostgreSQL aconteceu numa estação de trabalho com 767 MB RAM, processador Pentium IV com 1,5 Ghz e dois discos rígidos, um com 20 GB e outro, comportando as bases de dados com 180 GB. O Sistema Operacional foi o Kurumin Linux 4.2, baseado no *kernel* versão 2.6.8.1 e auxiliado pelo ambiente de desenvolvimento Kdvelop.

A obtenção de resultados experimentais realizada em [32] caracterizou-se pela execução de cargas de trabalho durante três intervalos: 30, 60 e 90 minutos. Para avaliar a eficácia do Agente de Benefícios, utilizando uma carga de trabalho derivada do *benchmark* TPC-H, elegeu-se um conjunto de consultas representativas e executou-se o lote, independentemente do fator tempo.

Antes de submeter o conjunto de consultas ao Agente de Benefícios, realizou-se um levantamento de índices criados pelo *toolkit* no mesmo conjunto. Quanto mais próximo do grupo de índices listado na Tabela 2.1 chegasse o agente, mais eficaz seria. Eventualmente, poderia até criar um índice não proposto pelo *toolkit*, que melhorasse ainda mais a execução de alguma consulta.

Consulta	Tabela	Campos	Chave
1	Lineitem	L_shipdate	
2	Partsupp	Ps_partkey	Estrangeira
	Supplier	S_suppkey	Primária
4	Orders	O_orderdate	
	Lineitem	L_orderkey	Estrangeira

PUC-Rio - Certificação Digital Nº 0410827/CA

Consulta	Tabela	Campos	Chave
10	Orders	O_orderdate	
	Lineitem	L_orderkey	Estrangeira
	Customer	C_custkey	Primária
17	Lineitem	L_partkey	Estrangeira
19	Part	P_partkey	Primária

Tabela 2.1: 6 consultas escolhidas e 8 índices propostos pelo *toolkit*.

Cada linha da Tabela 2.1 representa um índice. Pode-se constatar a presença de oito diferentes, dos quais apenas um (sobre a tabela **Orders**, campo **O_orderdate**) não está associado à chaves primárias ou estrangeiras.

Cada uma das seis consultas foi executada utilizando-se o comando **evaluate** em um ambiente desprovido de índices, ou seja, antes de cada execução eliminaram-se os índices que porventura existissem. Mesmo ajustando as consultas para que a seletividade fosse a melhor possível, as consultas 1, 10, 17 e 19 tiveram que ser executadas mais de uma vez. A Tabela 2.2 apresenta os índices criados pelo Agente de Benefícios por consulta e a quantidade de execuções do comando **evaluate** que foram necessárias.

Consulta	Tabela	Campos	Execuções
1	Lineitem	L_shipdate	7
2	Partsupp	Ps_partkey	1
	Supplier	S_suppkey	1
4	Orders	O_orderdate	1
	Lineitem	L_orderkey	
10	Orders	O_orderdate	2
	Lineitem	L_orderkey	2
	Customer	C_custkey	1
17	Lineitem	L_partkey	6
	Part	P_brand, P_container, P_partkey	1
19	Part	P_partkey	3

Tabela 2.2: 6 consultas escolhidas, 9 índices criados pelo agente e quantas vezes foi necessário executar com **evaluate** cada consulta para chegar às criações.

Analisando a Tabela 2.2 pode-se observar que apenas um índice não foi criado (sobre a tabela **Lineitem**, consulta 4), porém o mesmo índice aparece na

PUC-Rio - Certificação Digital Nº 0410827/CA

consulta 10. O fato do índice não ter sido criado sequer como hipotético pode sinalizar uma deficiência na Heurística de Escolha de Índices Candidatos, entretanto, este problema é minimizado graças ao fato do mesmo índice ter sido criado na consulta seguinte.

O fato das Consultas 10 e 4 compartilharem dois índices sinalizou a necessidade em observar o que aconteceria caso índices previamente criados não fossem destruídos antes da execução das consultas:

- Executando a Consulta 10 após a 4, ou seja, aproveitando o fato do índice sobre o campo **l_orderdate** já ter sido criado, o índice sobre o campo **c_custkey** foi construído na primeira execução, porém não fez menção a **l_orderkey**.
- Executando a Consulta 4 após a 10, ou seja, aproveitando os três índices previamente criados, nenhum índice adicional foi sugerido, já que os índices necessários à Consulta 4 já tinham sido criados.

Conclui-se, então, que faria diferença a ordem da execução das consultas que tivessem potenciais índices em comum. Este fato suscita questões interessantes que poderiam levar à concepção de heurísticas que descobrissem previamente a ordem de execução ótima de forma a minimizar o somatório dos custos das consultas envolvidas. Este assunto foge ao escopo da presente dissertação.

O índice a mais criado pelo Agente de Benefícios (sobre a tabela **Part**, consulta 17) e não sugerido pelo *toolkit* proporcionou ganhos notáveis. O custo caiu 62,62% e a sua duração obteve um ganho da ordem de 61,45%.

Uma vez verificada a capacidade do Agente de Benefícios em criar os mesmos índices propostos pelo *toolkit*, achou-se interessante comparar o conjunto de índices obtidos aos que seriam propostos por ferramentas de apoio disponibilizadas em dois grandes SGBDs comerciais, a saber, Microsoft SQL Server e Oracle. O estudo e as conclusões obtidas aparecem nas próximas duas seções.

2.6 Resultados Experimentais com SQL Server 2005

A segunda bateria de testes utilizou o SGBD Microsoft SQL Server 2005 na mesma estação descrita em 2.5, porém sob Sistema Operacional Windows

PUC-Rio - Certificação Digital Nº 0410827/CA

2000 Server. As mesmas seis consultas escolhidas em 2.5 foram submetidas à ferramenta *Database Tuning Advisor*, discutida em [3]. Criaram-se as mesmas tabelas, sem índices, os mesmos dados foram carregados e submeteu-se cada Consulta em separado. Não houve a necessidade de reiniciar o ambiente antes de cada execução, pois a ferramenta DTA apenas sugere a criação de índices, ficando a cargo do DBA aceitar, ou não, as recomendações.

A Tabela 2.3 revela, para cada consulta, os índices que seriam criados.

Consulta	**Tabela**	**Campos**
1	Lineitem	L_shipdate, L_returnflag, L_linestatus
2	Partsupp	Ps_partkey, Ps_suppkey, Ps_supplycost
	Part	P_partkey
4	Orders	O_orderdate, O_orderkey, O_orderpriority
	Lineitem	L_orderkey, L_commitDATE, L_receiptDATE
10	Orders	O_orderdate, O_custkey, O_orderkey
	Lineitem	L_orderkey, L_returnflag
	Customer	C_custkey, C_name, C_acctbal, C_phone, C_address, C_comment
17	Lineitem	L_partkey
19	Part	P_brand, P_container, P_size, P_partkey
	Lineitem	L_shipinstruct, L_partkey, L_shipmode, L_quantity

Tabela 2.3: 6 consultas escolhidas, 11 índices criados pelo SQL Server 2005.

Uma explicação para o fato de terem sido criados mais índices com SQL Server 2005 poderia ser o fato de serem criados automaticamente estatísticas e índices virtuais que seriam aproveitados nas execuções subsequêntes. Não há como impedir a criação de índices virtuais, nem forçar suas eliminações.

PUC-Rio - Certificação Digital Nº 0410827/CA

Deve-se ressaltar que não faria sentido executar cada consulta mais de uma vez, pois o trabalho da ferramenta não se baseia na atribuição gradativa de benefícios. Acontece primeiro uma análise do comando, para depois concluir as recomendações.

Um fato que chamou a atenção foi a não recomendação, na Consulta 17, do índice sobre a tabela **Part** (campos **P_brand, P_container, P_partkey**). De fato, uma vez criado, o Otimizador do SQL Server 2005 não somente aproveitou o índice, como também o registrou-se uma queda notável no custo da consulta, na ordem de 69,31%.

2.7 Resultados Experimentais com Oracle 10g

A bateria de testes utilizando o SGBD Oracle 10g aconteceu numa estação de trabalho com 512 MB RAM, processador Pentium IV com 1,5 Ghz e um disco rígido de 40 GB. O Sistema Operacional foi o Windows 2000 Server.

Assim como realizado com SQL Server 2005, as mesmas seis consultas foram submetidas a uma ferramenta de apoio, denominada *SQL Adjust Advisor*, disponibilizada no produto *Tuning Pack* do *Enterprise Manager*. Criaram-se as mesmas tabelas, sem índices, os mesmos dados foram carregados e submeteu-se cada Consulta em separado. Também não houve a necessidade de reiniciar o ambiente antes de cada execução, pois também trabalha-se com recomendações. Como não se trabalha com atribuições gradativas de benefícios, também não ocorreu mais de uma execução por consulta.

A Tabela 2.4 revela, para cada consulta, os índices sugeridos pelo Oracle 10g.

Consulta	Tabela	Campos
1	Lineitem	L_shipdate
2	Nenhum índice criado	
4	Orders	O_orderdate
	Lineitem	L_orderkey
10	Orders	O_orderdate
	Lineitem	L_orderkey

PUC-Rio - Certificação Digital Nº 0410827/CA

Consulta	Tabela	Campos
17	Lineitem	L_partkey
	Part	P_brand, P_container
19	Part	P_brand, P_size
	Part	P_brand, P_container, P_size
	Lineitem	L_partkey

Tabela 2.4: 6 consultas escolhidas, 10 índices criados pelo Oracle 10g.

Perceba-se que não há sugestões para a Consulta 2. Ao criar manualmente os índices propostos na Tabela 2.2 (Consulta 2), o otimizador Oracle utilizou apenas um (tabela **Partsupp,** campo **Ps_partkey**). A duração teve ligeiro decréscimo (7%) e o custo, menor ainda (4%).

Observe-se também a ausência do índice sobre a tabela **Customer** (campo **C_custkey),** para a Consulta 10. Ao re-submetê-la, após a criação manual do índice, o otimizador do Oracle preferiu ignorar o novo índice. O mesmo ocorreu ao criar manualmente oíndice sobre a tabela **Part,** campos **P_brand**, **P_container**, **P_partkey** (Consulta 17).

Algumas razões poderiam explicar os poucos índices propostos pelo Oracle:

i. Para que tome-se a decisão de criar um índice, as tabelas deveriam ser mais volumosas. Provavelmente, existe um método otimizado para realizar varreduras completas em tabelas (*full table scan*).
ii. As estatísticas e os índices virtuais bastariam para o otimizador solucionar as consultas.
iii. O critério de seletividade observado pelo Oracle talvez seja mais rigoroso, isto é, consultas ainda mais seletivas deveriam ter sido fornecidas, para que valesse a pena a criação de índices.

2.8 Comentários Finais

Este capítulo apresentou um breve panorama sobre o estado das pesquisas referentes à auto-sintonia, oferecendo mais detalhes sobre os trabalhos de pesquisa que criaram os elementos necessários ao desenvolvimento desta dissertação ([26] e [32]).

PUC-Rio - Certificação Digital Nº 0410827/CA

Partindo das discussões propostas em [32], constatou-se a eficácia da Heurística de Benefícios quando submetida a uma carga de trabalho com características diferentes das percebidas em [32] (OLAP, ao invés de OLTP). Isto ficou evidente quando o Agente de Benefícios foi capaz de criar os índices necessários à execução de consultas onerosas, inclusive proporcionando uma melhora, já que um índice inédito e vantajoso também foi criado. O valor deste novo índice foi destacado ao submeter as mesmas consultas ao SGBD Microsoft SQL Server 2005. Sua ferramenta de ajuste de desempenho automático (DTA – *Database Tuning Advisor*) não foi capaz de identificá-lo. Quando criado manualmente, a Consulta que o utilizava apresentou notável melhora.

Os mesmos testes aconteceram utilizando o SGBD Oracle 10g, porém foi proposto um conjunto menor de índices. Como os três testes aconteceram sob configurações de hardware e software distintas, achou-se por bem não comparar durações, somado ao fato da não criação de índices, exceto com PostgreSQL.

Idealmente, a Heurística de Benefícios deveria ser eficaz em quaisquer tipos de cargas de trabalho. [32] propõe, mas não efetivou, a eliminação de índices criados quando estes mostrarem-se contraproducentes. Esta funcionalidade é desejável em cargas de trabalho possuindo muitas transações com alto nível de alteração de dados.

No próximo capítulo serão discutidas extensões às idéias propostas em [32]: detalha-se a implementação de destruição automática de índices, propõe-se uma alternativa ao acompanhamento de índices criados e, finalmente, a bateria de testes presente em [32] é refeita para observar o impacto da nova funcionalidade implementada.

PUC-Rio - Certificação Digital Nº 0410827/CA