

Um Agente de Software para Criação de Índices no PostgreSQL

Marcos Antonio Vaz Salles¹, Sérgio Lifschitz¹

¹Departamento de Informática – Pontifícia Universidade Católica do Rio de Janeiro (PUC-Rio)
Rua Marquês de São Vicente, 225, Gávea – 22453-900 Rio de Janeiro, RJ

mvsalles@inf.puc-rio.br, sergio@inf.puc-rio.br

Abstract. *This paper briefly describes a prototype developed as part of a Masters thesis focused on autonomic index creation for database systems. The system is composed of a software agent that collects SQL commands submitted to the DBMS, analyzes which indexes are appropriate for those commands and automatically creates them. The analysis process uses server extensions that enable the creation of hypothetical indexes. The agent's implementation was done in C++ and integrated in the open source DBMS PostgreSQL. The server extensions made in PostgreSQL to simulate hypothetical index configurations were coded in C.*

Resumo. *Este artigo descreve brevemente um protótipo desenvolvido como parte de uma dissertação de mestrado focada na criação autônoma de índices em sistemas de bancos de dados. O sistema é composto de um agente de software que coleta comandos SQL processados pelo SGBD, analisa quais índices seriam adequados para estes comandos e os cria automaticamente. O processo de análise se utiliza de extensões codificadas no servidor para a criação de índices hipotéticos. A implementação do agente foi realizada em C++ e integrada no SGBD de código fonte aberto PostgreSQL. As extensões feitas no PostgreSQL para simulação de configurações hipotéticas de índices foram escritas em C.*

1. Introdução

A sintonia de bancos de dados (*database tuning*) é a atividade de tornar a execução de uma aplicação de bancos de dados mais rápida [Shasha and Bonnet, 2003]. Por mais rápida podemos entender que a aplicação terá maior vazão (*throughput*) em termos das transações que executa ou que algumas de suas transações terão menores tempos de resposta.

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

Neste artigo comentamos a implementação de uma arquitetura de agentes para a auto-sintonia de índices em um sistema de bancos de dados. A arquitetura propõe o uso de um agente de *software* embutido no SGBD que obtém os comandos processados pelo SGBD e decide quando criar índices adequados para estes comandos. Esta arquitetura faz parte de uma dissertação de mestrado recentemente aceita no Departamento de Informática da PUC-Rio [Salles, 2004].

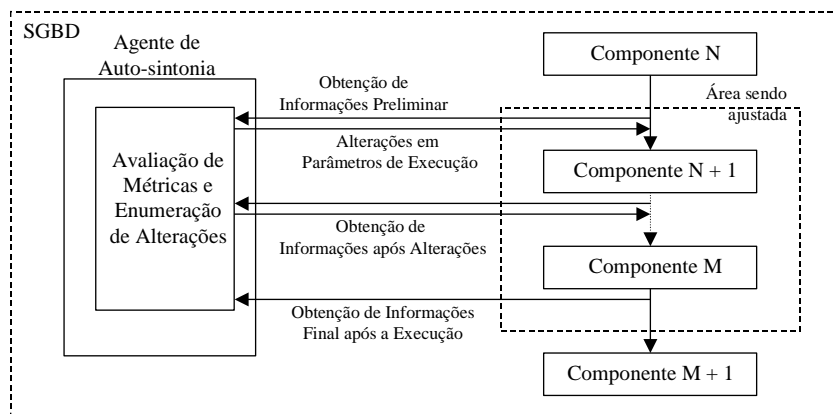


Figura 1: Modelo local para auto-sintonia usando agentes

O agente, denominado *Agente de Benefícios*, foi integrado com o SGBD de código fonte aberto PostgreSQL [Pos, 2004]. Foi necessário realizar extensões sobre este SGBD para permitir que o agente simulasse configurações hipotéticas de índices interagindo com o otimizador do sistema. Este tipo de técnica foi aplicada em outros trabalhos da literatura [Chaudhuri and Narasayya, 1997, Lohman et al., 2000], porém sempre em sistemas comerciais.

O restante deste artigo está organizado conforme descrito a seguir. A seção 2 descreve a arquitetura de integração do agente utilizado com o SGBD e discute qual a metodologia empregada para a tomada de decisões de auto-sintonia. Na seção 3, abordamos as demonstrações que podem ser realizadas do protótipo desenvolvido para a pesquisa. Por fim, na seção 4, apresentamos algumas conclusões e extensões possíveis do trabalho de implementação.

2. Arquitetura do Agente de Benefícios

Podemos classificar o foco das iniciativas de auto-sintonia de sistemas de bancos de dados em local ou global [Lifschitz et al., 2004]. Quando tratamos de uma iniciativa específica de sintonia, como é o nosso caso com a sintonia de índices, estamos preocupados eminentemente com o modelo local de auto-sintonia a ser empregado. A Figura 1 mostra este modelo para o *Agente de Benefícios*.

No modelo local, o agente interage com os componentes do SGBD referentes à sua área de ajuste. Esta interação é regida por um processo de auto-sintonia com as seguintes etapas:

- **Obtenção de Informações:** o agente se utiliza de sensores para receber informações e métricas dos componentes do SGBD.
- **Avaliação de Métricas:** a partir das informações coletadas, o agente estabelece crenças que dão suporte aos algoritmos utilizados para decidir se alguma ação de sintonia deve ser realizada.
- **Enumeração de Alterações Possíveis:** o agente aplica algoritmos ou heurísticas para enumerar alternativas de ajustes que podem levar a um melhor desempenho do sistema. Durante esta enumeração, o agente pode se utilizar de efetadores para simular cenários utilizando mecanismos previamente implementados nos componentes do SGBD.
- **Realização de Alterações:** os ajustes escolhidos pelos algoritmos ou heurísticas empregados pelo agente são aplicados aos componentes do sistema através do uso de efetadores.

No caso do *Agente de Benefícios*, a etapa de *Obtenção de Informações* é realizada através da coleta dos comandos SQL processados pelo otimizador. Em seguida, ocorre a *Avaliação de Métricas* com a atualização de crenças necessárias para a aplicação de heurísticas para seleção de índices. Em nosso trabalho, utilizamos uma heurística de [Lohman et al., 2000] para escolha de índices candidatos e propusemos uma nova heurística para determinação de quais índices deveriam ser materializados. Informações detalhadas sobre as heurísticas utilizadas para a tomada de decisões do agente estão fora do escopo deste artigo.

O agente aplica as heurísticas durante a etapa de *Enumeração de Alternativas Possíveis*. Neste momento, são criados índices hipotéticos¹ no sistema e o agente utiliza o otimizador para obter planos e custos estimados para a execução de comandos com estas configurações físicas simuladas. Por fim, é conduzida a *Realização de Alterações*, que é implementada pela criação ou remoção de índices reais do sistema de bancos de dados.

O processo de auto-sintonia apresentado se baseia em um *ciclo de controle de realimentação* para refinar progressivamente as decisões de sintonia tomadas com o conhecimento das métricas locais obtidas a partir dos componentes do SGBD. O uso deste tipo de abordagem para processos de auto-sintonia é comum na literatura, conforme é detalhado em [Lifschitz et al., 2004].

3. Características da Demonstração

O protótipo que foi desenvolvido para apoiar nossa pesquisa permite que realizemos duas demonstrações de funcionalidade. Na primeira, mostramos como o servidor PostgreSQL foi estendido para permitir a simulação de índices hipotéticos. Este mecanismo foi implementado de forma a permitir que a simulação seja executada tanto de forma automática como por um DBA que interaja com o sistema. As extensões do servidor foram implementadas em C.

Na segunda demonstração, ilustramos como o agente adapta os índices do sistema aos comandos SQL que são submetidos. Fazemos a execução de uma carga transacional com múltiplos usuários e verificamos como o uso do agente influi na vazão do sistema. O agente foi escrito em C++ e integrado com o SGBD PostgreSQL.

3.1. Demonstração das Extensões do Servidor

Em nossa implementação, estendemos a linguagem de definição de dados do SGBD PostgreSQL para conter os seguintes novos comandos:

1. *create hypothetical index*;
2. *drop hypothetical index*;
3. *explain hypothetical*;

Os primeiros dois comandos têm como finalidade possibilitar o registro e a remoção de índices hipotéticos. A exemplo de [Lohman et al., 2000], estas definições são armazenadas no próprio catálogo, que foi estendido para permitir diferenciar índices hipotéticos de índices reais. Já o terceiro comando permite que sejam obtidos planos de execução e custos de consultas levando em consideração os índices hipotéticos definidos.

Para ilustrar o comportamento dos novos comandos adicionados ao sistema, apresentamos um exemplo. Suponha que estejamos escrevendo uma consulta sobre uma base de dados simples de vendas. Podemos ver o plano que o SGBD escolheria para a consulta utilizando o comando *explain*:

¹Índices hipotéticos são utilizados para avaliar a qualidade de configurações simuladas de índices. Estes tipos de índices existem apenas no catálogo do sistema e não possuem extensão física.

```

simple=# explain
simple-# select prodNum, data, sum(valor) as total
simple-# from venda
simple-# where valor > 1500000 and
simple-#         data between '20040101' and '20040131'
simple-# group by prodNum, data;

```

QUERY PLAN

```

-----
HashAggregate (cost=25388.79..25388.83 rows=12 width=21)
-> Seq Scan on venda (cost=0.00..25367.00 rows=2906
      width=21)
      Filter: ((valor > 1500000::numeric) AND
              (data >= '2004-01-01'::date) AND
              (data <= '2004-01-31'::date))
(3 rows)

```

No plano, podemos perceber que o SGBD fará uma varredura seqüencial sobre a tabela de vendas e aplicará os predicados sobre as colunas de valor e de data. A quantidade de linhas esperada como resultado é de 2906 e o custo de processamento da varredura será de 25367.00. Após a varredura, um operador de agregação será aplicado para processar a cláusula *group by* do comando. O custo total esperado é de 25388.83.

Gostaríamos de verificar a utilidade de um determinado índice sobre a tabela de vendas para nossa consulta. Podemos criar um índice hipotético conforme mostrado abaixo:

```

simple=# create hypothetical index hi_venda_valor_data
simple-# on venda (valor, data);

```

Reparar que, ao fazermos isto, o índice não é materializado e, assim, o processamento de outras transações no sistema continua praticamente inafetado. Podemos, agora, verificar que plano seria escolhido pelo SGBD se este índice fosse materializado na base. Utilizamos o comando *explain hypothetical*:

```

simple=# explain hypothetical
simple-# select prodNum, data, sum(valor) as total
simple-# from venda
simple-# where valor > 1500000 and
simple-#         data between '20040101' and '20040131'
simple-# group by prodNum, data;

```

QUERY PLAN

```

-----
HashAggregate (cost=10514.63..10514.66 rows=12 width=21)
-> Index Scan using hi_venda_valor_data on venda
      (cost=0.00..10492.83 rows= 2906 width=21)
      Index Cond: ((valor > 1500000::numeric) AND
                  (data >= '2004-01-01'::date) AND
                  (data <= '2004-01-31'::date))
(3 rows)

```

Podemos perceber que, caso existisse, o índice sobre as colunas de valor e data seria escolhido para o processamento da consulta. A quantidade esperada de linhas a

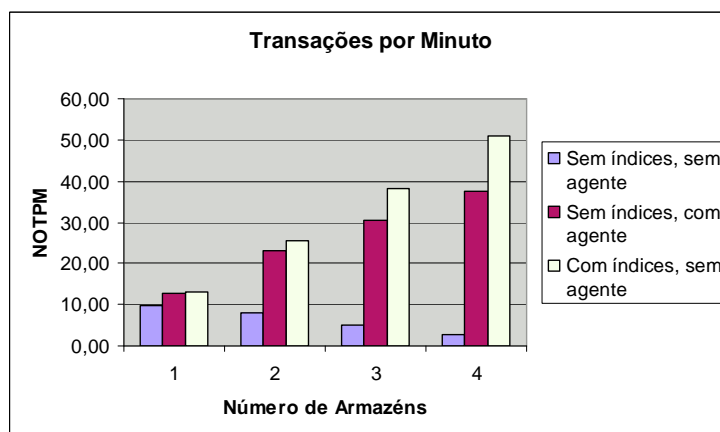


Figura 2: Vazão observada para o agente em testes de 90 min

ser retornada pela varredura indexada seria de 2906 a um custo de 10492.83. Após a varredura indexada, seria aplicado o operador de agregação e o plano como um todo teria um custo de 10514.66. Assim, o novo índice traria benefícios à consulta que pretendemos executar.

3.2. Demonstração do Agente

Os testes de desempenho da implementação do agente foram realizados com o *toolkit* OSDL DBT-2 [OSDL-DBT2, 2004]. Este simula a execução de uma carga transacional, inspirada no *benchmark* TPC-C, com múltiplos usuários no SGBD. Ocorre a simulação das operações de um varejista que trabalha com diversos armazéns para estocagem de seus produtos.

O *toolkit* já vem com um conjunto de índices sugeridos para o melhor processamento da carga de trabalho. Alguns destes índices são criados como consequência da criação de chaves primárias. Para avaliarmos a contribuição trazida pelo nosso agente à vazão observada nos testes, estabelecemos três cenários:

1. Execução da carga de trabalho em uma base sem índices e com o agente desligado.
2. Execução da carga de trabalho em uma base sem índices e com o agente ligado.
3. Execução da carga de trabalho em uma base com os índices sugeridos pelo *toolkit* e com o agente desligado.

Existem dois parâmetros que modificam as características de cada teste: a quantidade de armazéns configurados para o *toolkit* e o tempo de duração. Quanto mais armazéns, maior é a carga efetivamente submetida ao SGBD, uma vez que temos bases de dados maiores e também mais terminais acessando o sistema de forma concorrente.

O tempo de duração afeta a vazão observada uma vez que o agente precisa de um período de aprendizado para determinar os índices adequados para a carga de trabalho. Na Figura 2, mostramos as vazões observadas em um teste com duração de 90 minutos para os três cenários descritos anteriormente.

Temos uma vazão decrescente para a execução da carga de trabalho em uma base sem índices e sem o agente ligado. Neste cenário, os tempos de resposta aumentam, uma vez que processamos as consultas através de varreduras sequenciais em bases de dados de crescente tamanho. Já no terceiro cenário, em que utilizamos os índices do *toolkit*, temos o comportamento inverso. A vazão do sistema cresce à medida que os tamanhos da base de dados aumentam. Isto se deve ao fato de os tempos de resposta permanecerem praticamente constantes e de o sistema possuir mais terminais simultâneos submetendo transações quando configuramos mais armazéns para o *toolkit*.

A vazão observada para o cenário com o agente se situa entre as vazões dos outros dois cenários. O agente passa por um período de aprendizado, em que ocorre a criação de diversos índices que podem trazer benefícios para os comandos SQL da carga de trabalho. Uma vez que os índices indicados são criados, o agente permanece monitorando o sistema para verificar se alguma nova intervenção é necessária. Em [Salles, 2004], diversos outros tipos de testes são explorados com a implementação realizada.

Na demonstração, pretendemos executar o agente com a carga de trabalho do *toolkit* OSDL DBT-2 durante um curto período. Observaremos a vazão e os índices que são escolhidos pelo agente para o sistema.

4. Conclusões

Implementamos um protótipo para validar uma das arquiteturas para auto-sintonia de índices propostas em [Salles, 2004]. Este protótipo integra um agente de *software* ao SGBD PostgreSQL. Possui como característica, ainda, um conjunto de extensões realizadas sobre o servidor de bancos de dados que permitem a simulação de configurações hipotéticas de índices.

O protótipo pode ser estendido em alguns pontos. Seria interessante permitir que o agente realizasse a remoção de índices previamente existentes na base de dados de forma automática. Outro ponto interessante é a investigação de diferentes heurísticas de decisão para o agente.

Referências

- Chaudhuri, S. and Narasayya, V. (1997). An efficient, cost-driven index selection tool for microsoft sqlserver. In *Proceedings of the International Conference on Very Large Databases (VLDB)*, pages 146–155.
- Lifschitz, S., Milanés, A. Y., and Salles, M. A. V. (2004). Estado da arte em auto-sintonia de sistemas de bancos de dados relacionais. Technical report, Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro (PUC-Rio).
- Lohman, G., Valentin, G., Zilio, D., Zuliani, M., and Skelley, A. (2000). DB2 advisor: An optimizer smart enough to recommend its own indexes. In *Proceedings of the IEEE International Conference on Data Engineering (ICDE)*, pages 101–110.
- OSDL-DBT2 (2004). Open source development labs database test 2 (osdl-dbt-2). http://www.osdl.org/lab_activities/kernel_testing/osdl_database_test_suite/osdl_dbt-2/.
- Pos (2004). PostgreSQL. <http://www.postgresql.org>.
- Salles, M. A. V. (2004). Criação autônoma de Índices em bancos de dados. Master's thesis, Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro (PUC-Rio). orientada por Sérgio Lifschitz e aceita pela Comissão Julgadora em 15/07/2004.
- Shasha, D. and Bonnet, P. (2003). *Database Tuning: Principles, Experiments and Troubleshooting Techniques*. Morgan Kaufmann.