Particionamento como Ação de Sintonia Fina em Bancos de Dados Relacionais

Ana Carolina Almeida[1], Antony Seabra de Medeiros[2], Rogério Luís de Carvalho Costa[2], Sérgio Lifschitz[2]

[1]Depto de Informática e Ciência da Computação UERJ

[2]Departamento de Informática PUC-Rio

ana.almeida@ime.uerj.br, {amedeiros, rogcosta, sergio}@inf.puc-rio.br

Abstract. *Table partitioning is usually applied for distributing data. It may also be considered for performance issues, since it allows full scans involving only those partitions that satisfy the queries predicates. This work presents heuristics that select a partitioning strategy for each table and estimate their workload benefits. We also present some studies on the use of our proposed heuristics with the TPC-H benchmark. We show that partitioning can be effectively used as a first-order database tuning technique.*

Resumo. *O particionamento é uma técnica utilizada para distribuição de dados que também possui potencial para ajuste de desempenho, pois permite que a varredura das tabelas seja realizada somente nas partições que satisfazem os predicados das consultas. Este trabalho apresenta heurísticas que permitem selecionar uma estratégia de particionamento para cada tabela e estimar seus benefícios para a carga de trabalho. Aplica-se as heurísticas com o benchmark TPC-H e mostra-se que o particionamento pode ser utilizado como uma ação de sintonia fina de primeira ordem.*

1 – Introdução

A sintonia fina de bancos de dados consiste na realização de alterações visando obter melhor desempenho para a execução de comandos de bancos de dados. As ações de sintonia fina podem reduzir de forma significativa o tempo de execução dos comandos, fazendo, por exemplo, com que comandos e scripts que levariam vários minutos para serem concluídos passem a ser executados em apenas alguns poucos segundos.

Dentre as estratégias de sintonia fina mais comumente utilizadas por administradores de bancos de dados, estão a construção de estruturas de acesso, tais como índices, índices parciais e visões materializadas, a desnormalização e a reescrita de consultas. Por outro lado, o particionamento de tabelas consiste na transformação de uma tabela em dois ou mais fragmentos, onde cada fragmento é, na verdade, um subconjunto dos dados da tabela (linhas ou colunas) gerado com base em critério pré-definido.

Neste artigo, estudamos o uso do particionamento como ação de sintonia fina. A seleção dos critérios de fragmentação é uma das maiores dificuldades no uso do particionamento. Propomos heurísticas que permitem selecionar estratégias de

particionamento com base nos benefícios gerados em termos da redução do tempo de execução de uma carga de trabalho. Realizamos, também, diversos estudos que permitem validar a aplicação das heurísticas propostas e observar o uso do particionamento como ação de sintonia fina.

Na próxima seção, relembramos alguns conceitos fundamentais relacionados ao particionamento de tabelas e mencionamos alguns trabalhos relacionados relevantes. Na Seção 3 apresentamos as heurísticas propostas. A Seção 4 apresenta as avaliações experimentais realizadas e os resultados obtidos. Finalmente, a Seção 5 traz algumas conclusões e comentários finais.

2 – Fundamentos e Trabalhos Relacionados

O particionamento (ou fragmentação) de uma tabela pode ser horizontal, vertical ou híbrido. No particionamento horizontal, divide-se a tabela em subconjuntos disjuntos de linhas ou tuplas (Valduriez, 2011), sendo cada subconjunto (ou partição) definido com base em um critério, composto por um ou mais atributos e os respectivos valores para cada partição. A fragmentação horizontal será denominada *primária* se a chave do particionamento for um atributo da tabela que está sendo particionada ou *derivada* no caso contrário. No particionamento vertical, divide-se a tabela em subconjuntos de colunas, repetindo-se sua chave primária em cada partição. Por fim, no particionamento híbrido, combina-se o particionamento horizontal e o vertical.

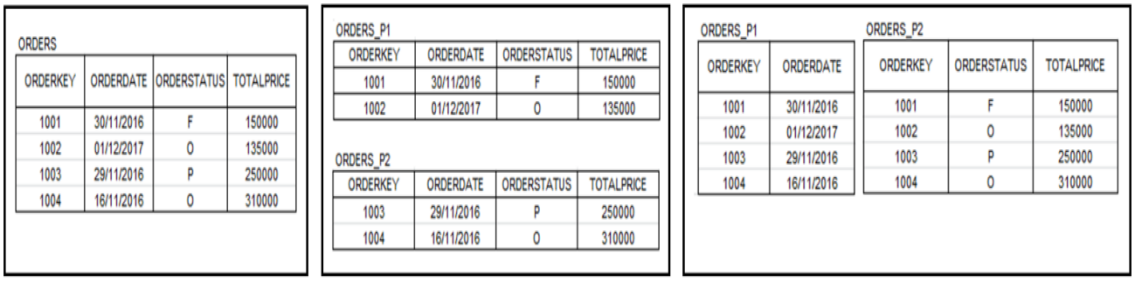

Figura 1: Particionamento horizontal e vertical

Na Figura 1, a tabela *orders* possui 4 tuplas. No particionamento horizontal, por exemplo, usando *totalprice* como chave do particionamento, pode-se ter duas partições: a partição *orders_p1* contendo as 2 tuplas com *totalprice<200000* e a partição *orders_p2* contendo as 2 tuplas com *totalprice≥200000*. Já no particionamento vertical, por exemplo, a partição *orders_p1* contém as colunas *orderkey* e *orderdate* das 4 tuplas, enquanto a partição *orders_p2* contém as colunas *orderkey*, *orderstatus* e *totalprice* das 4 tuplas.

Ambos os tipos de particionamento possuem potencial para sintonia fina na medida em que podem reduzir a quantidade de operações de entrada e saída (I/O) a serem realizadas durante a execução de comandos SQL. Por exemplo, para a consulta *Select * From orders Where totalprice>=250000*, somente a partição *orders_p2* seria acessada, ao invés de toda a tabela *orders*, diminuindo o número de páginas a serem percorridas e, assim, reduzindo o custo de execução da consulta.

A redução do custo de acesso a dados é uma preocupação constante em grandes bases de dados. Para tal, utilizam-se técnicas de *sintonia fina*. Em (Monteiro, 2008), a sintonia fina automática é apresentada, incluindo a seleção automática de índices, visões materializadas, índices sobre visões e particionamento. O trabalho enfatiza que a manutenção automática do projeto físico deve conter a coleta da carga de trabalho, a seleção das estruturas apropriadas e a manutenção dessas estruturas. É apresentada a heurística HISAI e uma ferramenta para apoiar o uso da manutenção automática de índices, que pode ser utilizada com vários SGBDs padrão de mercado. A heurística apresenta um modelo para estimar o benefício de um índice como ação de sintonia fina.

Em (Almeida et. al, 2015) é apresentado DBX, ferramenta para a manutenção automática do projeto físico em bancos de dados relacionais. DBX é, de fato, um framework, dado que pode ser instanciado para atividades distintas de sintonia fina envolvendo visões materializadas e índices, e, potencialmente, outras ações. DBX é uma ferramenta não-intrusiva, independente de SGBD, que pode ser executada de forma contínua e sem intervenção de um DBA. As ações específicas de sintonia fina automática são determinadas em função da análise dos planos de execução capturados e possíveis modificações em planos hipotéticos, com implementação da heurística HISAI (Monteiro, 2008).

Existem ainda trabalhos na literatura que exibem o potencial de benefício do particionamento como ação de sintonia fina, como (Agrawal, 2004) e (Bellatreche, 2004). Outros, mais focados em selecionar estratégias de particionamento, como (Curino, 2010), utilizam grafos para representar uma carga de trabalho e algoritmos de particionamento de grafos para selecionar uma estratégia de particionamento para as tabelas envolvidas. Há também alguns que, como (Wang, 2013), utilizam a análise das consultas e atributos das tabelas para determinar uma estratégia de particionamento.

3 – Uso do particionamento para sintonia fina

Neste trabalho, propomos a heurística HISAP – *Heurística Integrada para Seleção e Acompanhamento de Partições* – cujo objetivo é analisar todas as consultas e atualizações da carga de trabalho e selecionar uma estratégia de particionamento para o esquema do banco de dados, com base no benefício obtido nos custos dos planos de execução correspondentes às consultas e atualizações.

3.1 – Seleção da estratégia de particionamento

A estratégia proposta é composta de seis passos, descritos a seguir:

Passo 1 – Inicialmente, a heurística constrói uma lista de predicados simples utilizados nas consultas e, para cada predicado, mantém o custo da operação, que vai sendo acumulado durante a execução da heurística, um contador para registrar a frequência de acesso, incrementado a cada consulta que utiliza este predicado, e outro contador para a frequência de atualização, incrementado a cada atualização que utiliza este predicado.

Note-se que os predicados são decompostos independentemente do tipo de operador utilizado no plano de execução. Na heurística HISAI importa analisar os predicados de operações *Sequential Scan*. São eles os candidatos a chaves de índices. Em HISAP, caso mantida a mesma restrição, a heurística estaria propondo o particionamento como

estratégia secundária, para os casos onde os índices não seriam considerados benéficos a ponto de deixarem de ser hipotéticos. Todas as operações são analisadas, incluindo *Index Scan* e *Index Seek*, porque é possível que predicados para os quais o otimizador selecione um índice sejam chaves de particionamento candidatas e de fato ofereçam um benefício maior com tal ação de sintonia fina.

Passo 2 – Com base nesta lista de predicados, uma lista de chaves candidatas é construída, acumulando-se o custo, a frequência de acesso e a frequência de atualização. Para cada chave candidata, o valor *(access frequency – update frequency)* é calculado e, para todos os valores negativos, a entrada correspondente é removida da lista.

Passo 3 – Em seguida, a lista é classificada em ordem decrescente de custo, o que garante tratar primeiramente os predicados com maior custo para a varredura das tabelas. A lista é percorrida e, para cada chave candidata, as subfases da seleção da estratégia de particionamento são executadas, onde são definidos a chave e o o tipo de particionamento, o número de partições e o valor da chave para cada partição.

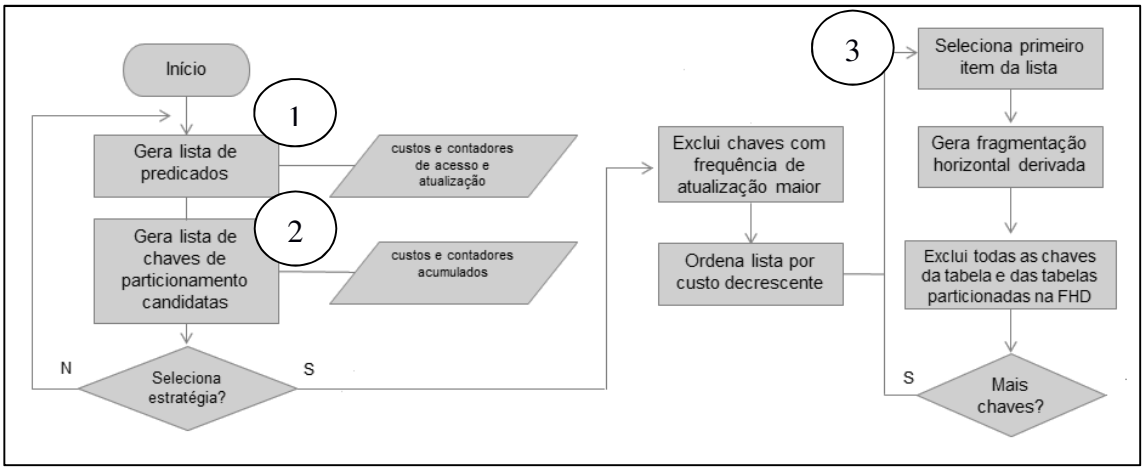

Figura 2: Fluxograma da heurística HISAP

Quando não há mais chaves de particionamento a serem definidas, termina-se o ciclo e inicia-se a definição dos tipos e dos valores da chave em cada partição de cada tabela.

Passo 4 – O tipo de particionamento é definido com base no tipo da coluna e no tipo de acesso mais frequente sobre a coluna. Tipos numéricos e data cujos predicados usam faixas de valores convergem para o tipo *range*. Tipos compostos por caracteres cujos predicados usam valores discretos convergem para o tipo *list*. E tipos numéricos usados em chaves de tabelas convergem para o tipo *hash*.

Passo 5 – As partições da tabela e os valores da chave para cada partição são gerados a partir do histograma da coluna. A heurística aceita dois parâmetros de entrada: o número máximo de partições e o tipo de balanceamento das partições – baseado em volume ou carga de trabalho. Os balanceamentos não são garantidos pela heurística, mas sim pela sua implementação. Por exemplo, um balanceamento de carga pode ser alcançado com o uso do algoritmo BEA (*Bond Energy Algorithm*), que faz uso de uma matriz de afinidades para gerar novos grupos com afinidades maiores. Em (Valduriez, 2011), o algoritmo é utilizado para o particionamento vertical. Uma matriz de afinidades

é obtida substituindo-se as *colunas* referenciadas nas consultas pelas *partições* acessadas nas consultas.

Passo 6 – Ao gerar o particionamento para uma tabela, são gerados os particionamentos das tabelas que compõem a fragmentação horizontal derivada. Assim, as chaves candidatas de todas as tabelas consideradas nesta operação são removidas da lista. Em seguida, inicia-se um novo ciclo que analisa novos subconjuntos de tabelas do esquema do banco de dados. O processo se repete até o término da lista (para particionar todas as tabelas do banco de dados) ou até que seja alcançado um limiar (baseado em um fator de corte) que pode ser definido pelo DBA.

3.2 – Estimativa de benefício do particionamento

Seja *W* uma carga de trabalho submetida a um banco de dados e $B(Q_i)$ o benefício de um particionamento para a consulta $Qi \in W$. Para uma dada consulta *Qi* da carga de trabalho, seu plano de execução utiliza operadores de varredura em estruturas de acesso, a saber, uma tabela ou um índice. Quando um índice não é utilizado, tem-se um operador do tipo *Sequential Scan* no plano de execução. Nota-se que quando uma tabela é particionada, a varredura ocorre somente na(s) partição(ões) da tabela que satisfaz(em) o predicado. Logo, o benefício $B(Q_i)$ de um particionamento para uma consulta assim formada pode ser calculado pela diferença entre o custo do *Sequential Scan* na tabela inteira e o custo do *Sequential Scan* na tabela particionada (Medeiros, 2017).

Por outro lado, quando um índice é utilizado, tem-se um operador do tipo *Index Scan* (ou *Index Seek*) no plano de execução. O benefício $B(Q_i)$ de um particionamento para uma consulta assim formada poderia ser calculado pela diferença entre o custo do *Index Scan* (ou *Index Seek*) no índice não-particionado e o custo do *Index Scan* (ou *Index Seek*) no índice particionado. Entretanto, como as seletividades dos predicados nesta tabela poderão sofrer uma diminuição, considera-se que o índice não será utilizado e, portanto, o custo é calculado pela diferença entre o custo do *Index Scan* (ou *Index Seek*) no índice não-particionado e o custo do *Sequential Scan* na tabela particionada.

Seja $C(SS)$ o custo do *Sequential Scan* em uma tabela, $C(IS)$ o custo do *Index Scan* (ou *Index Seek*) em um índice e $C(SSP_k)$ o custo do *Sequential Scan* em uma tabela hipoteticamente particionada através da chave de particionamento P_k.

Considerando $n=1..t$ o número de varreduras contidas no plano de execução de Q_i, o benefício $B(Q_i)$ é o somatório dos benefícios alcançados para cada varredura:

```
SE operador = Sequential Scan
    B(Qi) = B(Qi) + ∑ n=1..t ( C(SSn) – C(SSPkn) )
SENÃO
    B(Qi) = B(Qi) + ∑ n=1..t ( C(ISn) - C(SSPkn) )
FIM-SE
```

Considerando a criação de partições balanceadas, isto é, com uma distribuição uniforme do volume de dados em cada uma delas, efetuar uma varredura em uma estrutura em uma única partição significa percorrer o número total de linhas da tabela dividido pelo número total de partições.

Seja NP_k o número de partições gerado pelo particionamento de uma tabela usando a chave P_k e $NP(SSP_k)$ o número de partições acessadas em uma busca na estrutura particionada. Logo, o custo do *Sequential Scan* em uma estrutura particionada, *C(SSPk)*, pode ser assim estimado:

$$C(SSP_k) = (C(SS) / NP_k) * NP(SSP_k)$$

Da mesma forma, o custo do *Index Scan* num índice, *C(ISPk)*, poderia ser estimado como:

$$C(ISP_k) = (C(IS) / NP_k) * NP(ISP_k)$$

Ao analisar uma carga de trabalho, o benefício acumulado de um particionamento hipotético *AB(W)* será, portanto, o somatório dos benefícios alcançados em cada consulta da carga de trabalho multiplicados pela frequência de cada consulta.

$$AB(W) = \sum(B(Q_i) \times F(Q_i))$$

Duas considerações fundamentais são necessárias na implementação do particionamento hipotético. Primeiro, a fragmentação horizontal derivada gera um particionamento em cadeia, que pode melhorar o desempenho das consultas que usam junção nessas tabelas e o benefício gerado pelo particionamento. Quando um particionamento hipotético é criado para uma determinada tabela, um particionamento hipotético para todas as tabelas que fazem parte da cadeia deve ser criado também. Como exemplo, no banco de dados do TPC-H, o particionamento da tabela *orders* leva ao particionamento da tabela *lineitem*. Isto pode garantir que numa consulta com junção entre as duas tabelas, não somente as partições das tabelas que satisfazem os predicados sejam acessadas, mas também que somente as partições-filhas das partições-mães sejam acessadas. Segundo, o particionamento não somente altera os volumes de dados a serem percorridos pelos operadores de um plano de execução, mas pode influenciar as decisões do otimizador na medida em que tais operadores são escolhidos com base nesses volumes de dados e nas estatísticas correspondentes. Ao alterar estratégias de junção, por exemplo, o benefício do particionamento deixa de ser tão somente a diferença no custo da varredura (Medeiros, 2017).

Com bases nestas duas considerações, o benefício do particionamento para uma consulta e, portanto, para a carga de trabalho, pode sofrer variações. Estas variações dependem fundamentalmente da implementação do particionamento no SGBD e pode ser necessário fazer ajustes nas estimativas.

4 – Avaliações experimentais

A seguir apresentam-se os resultados encontrados nas simulações de execução da heurística HISAP. Utilizou-se o *benchmark* de referência TPC-H (TPC, 2017). As bases de dados, com volumes de 1GB, 10GB e 50GB, foram instaladas sob o SGBD *PostgreSQL 9.6.1* sob a configuração padrão em um servidor Intel Core i3 com 4 GB de memória RAM e HD de 500 GB, rodando sistema operacional Windows 7.

Foram criados três bancos de dados, *tpch*, *tpch-i* e *tpch-p*, para coletar os resultados das consultas com, respectivamente, nenhuma estratégia de sintonia fina, índices como ação de sintonia fina e particionamento como ação de sintonia fina. Foram utilizados 3 cenários de teste com características distintas para avaliação das heurísticas de seleção da estratégia de particionamento. Em todos os cenários são utilizadas as consultas fornecidas pelo próprio *benchmark*, submetidas aos SGBDs através de *scripts Powershell*. Considerando as características de um *data warehouse* típico com dados de 5 anos, optamos por definir o número máximo de partições em *cinco* e o tipo de balanceamento *por volume*. A métrica usada para avaliação dos resultados é o tempo de execução da consulta. A contagem do tempo (em milissegundos) é realizada pelo próprio SGBD e o tempo de execução de cada consulta apresentada a seguir é a média de três execuções, a partir da segunda execução, ou seja, a primeira execução é descartada.

4.1 – Cenário 1 – Particionamento, índices completos e índices parciais

Este cenário de teste é composto pela execução de variações da consulta 1 (Figura 3) e objetiva estudar o comportamento das estratégias de sintonia fina quando variamos a seletividade dos predicados em uma consulta sem joins, CPU-bound.

```
SELECT L_RETURNFLAG, L_LINESTATUS, SUM(L_QUANTITY) AS SUM_QTY,
SUM(L_EXTENDEDPRICE) AS SUM_BASE_PRICE,
SUM(L_EXTENDEDPRICE*(1-L_DISCOUNT)) AS SUM_DISC_PRICE,
SUM(L_EXTENDEDPRICE*(1-L_DISCOUNT)*(1+L_TAX)) AS SUM_CHARGE,
AVG(L_QUANTITY) AS AVG_QTY, AVG(L_EXTENDEDPRICE) AS AVG_PRICE,
AVG(L_DISCOUNT) AS AVG_DISC, COUNT(*) AS COUNT_ORDER
FROM LINEITEM WHERE L_SHIPDATE <= '1993-01-01'
GROUP BY L_RETURNFLAG, L_LINESTATUS
ORDER BY L_RETURNFLAG, L_LINESTATUS
```

Figura 3: Consulta 1

O maior custo nos planos de execução associados às variações desta consulta está no *Sequential Scan* da tabela *lineitem*. Uma ação de sintonia fina (sugerida por HISAI) é a criação de um índice *btree* nas colunas (*l_shipdate*, *l_returnflag*, *l_linestatus*), justamente as colunas que aparecem nas cláusulas *Where* e *Group By*. O otimizador seleciona este índice quando a seletividade é suficientemente alta (*tpch-i*). Na execução da heurística HISAP, as chaves de particionamento candidatas são *l_shipdate*, *l_returnflag* e *l_linestatus*, sendo que *l_shipdate* possui o maior custo associado. Como a coluna é do tipo *date* e as consultas utilizam predicados com faixas de valores, o tipo do particionamento é *range.* Através do histograma da coluna, são geradas cinco partições utilizando o balanceamento por volume.

Quando a consulta é satisfeita em *tpch-p* com o acesso de uma única partição de *orders*, temos os resultados mostrados no gráfico da Figura 4. Além disso, quando temos um índice parcial que contemple o mesmo predicado satisfeito com o acesso a uma única partição, temos um desempenho (*tpch-pi*) semelhante à estratégia de particionamento.

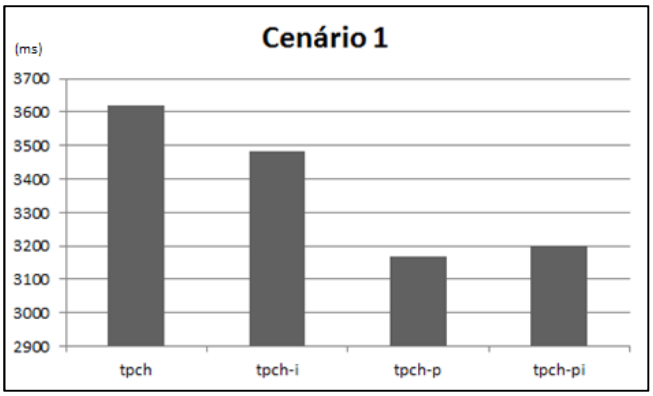

Figura 4: Cenário 1 – resultados

Quanto mais abrangente for a faixa de valores contemplada pelo predicado da consulta, maior o número de partições a serem acessadas e, consequentemente, maiores o custo da consulta e o tempo necessário para executá-la. Por exemplo, com o filtro *Where l_shipdate ≤ '1996-01-01'*, o *Sequential Scan* é executado sobre as partições que contém as linhas correspondentes, neste caso 3 partições. Por outro lado, ao aumentar-se a faixa de valores do predicado, sua seletividade é diminuída, o que acarreta diretamente na diminuição da probabilidade do otimizador selecionar um índice para seu plano de execução. O plano de execução desta mesma consulta no banco de dados *tpch-i* não seleciona o índice anteriormente utilizado nas colunas (*l_shipdate, l_returnflag, l_linestatus*). Ao contrário, observa-se um *Sequential Scan* na tabela inteira. Portanto, o crescimento do número de partições acessadas por uma consulta está associado à diminuição da seletividade do predicado e, por consequência direta, a diminuição da probabilidade do uso de índices.

Forçamos a varredura em n=1, 2, 3, 4 e 5 partições da tabela *lineitem* e estudamos o comportamento das estratégias de sintonia fina. Em n=1 o custo em C_P (configuração de particionamento em HISAP) é bem menor. Para n=2, com o aumento da faixa de valores que resulta no acesso a duas partições da tabela, o custo em C_I é quase igual ao aferido em C, sendo o custo em C_P ainda bem menor. Para n=3 e n=4, o índice já não é mais selecionado e C_P se mantém menor que C, resultado do acesso a 3 e 4 partições, respectivamente. Para n=5, todas as partições são acessadas e os custos são similares, conforme mostrado na Figura 5.

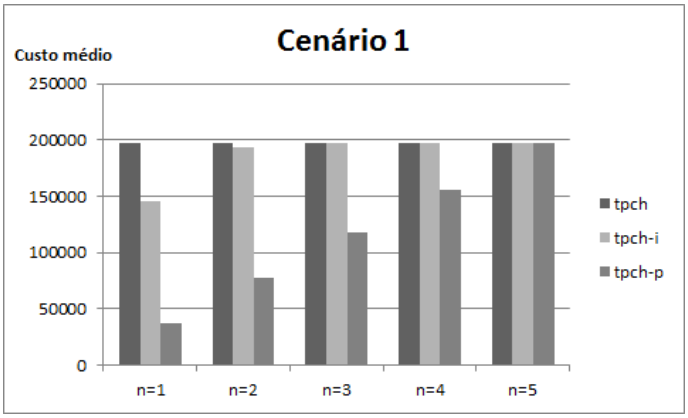

Figura 5: Cenário 1 – comportamento das estratégias de sintonia fina

4.2 – Cenário 2 – Benefício do particionamento para a carga de trabalho

O cenário 2 é composto pela execução de variações das consultas 3 e 8 e objetiva mostrar o benefício do particionamento em consultas I/O-bound. A consulta 3 (Figura 6) utiliza dois predicados nas colunas *o_orderdate* e *l_shipdate*. Ainda, incrementa a frequência de acesso para *l_orderkey*, *o_orderdate* e *o_shippriority* na cláusula *Group By* e incrementa mais uma vez a frequência de *o_orderdate* na cláusula *Order By*.

```
SELECT L_ORDERKEY, SUM(L_EXTENDEDPRICE*(1-L_DISCOUNT)) AS REVENUE,
O_ORDERDATE, O_SHIPPRIORITY
FROM CUSTOMER, ORDERS, LINEITEM
WHERE C_MKTSEGMENT = 'BUILDING' AND C_CUSTKEY = O_CUSTKEY
AND L_ORDERKEY = O_ORDERKEY AND O_ORDERDATE < '1994-03-15'
AND L_SHIPDATE > '1999-03-15'
GROUP BY L_ORDERKEY, O_ORDERDATE, O_SHIPPRIORITY
ORDER BY REVENUE DESC, O_ORDERDATE LIMIT 10
```

Figura 6: Consulta 3

Seu plano de execução mostra, inicialmente, um *Sequential Scan* em *customer* e um *Sequential Scan* em *orders*, e uma junção usando *(orders.o_custkey = customer.c_custkey)*. O maior custo está associado ao *Sequential Scan* da tabela *orders.* Em *tpch-i*, o otimizador seleciona os índices *i_orders_orderdate* e *i_lineitem_shipdate*; assim, um *Index Scan* é realizado em ambos os índices, fazendo com que o custo deste plano seja menor que o custo do plano gerado em *tpch*.

A consulta 8 (Figura 7) em *tpch* é resolvida com varreduras sequenciais e a aplicação dos filtros apropriados durante as varreduras. As linhas de *orders* e *lineitem* são unidas através de um *Nested Loop Join* e o maior custo no plano de execução está relacionado ao *Sequential Scan* de *orders*. Em *tpch-i* o índice *i_orders_orderdate* é selecionado para a varredura de *orders*, enquanto para *lineitem* a varredura sequencial é preferida. O maior custo entre os predicados em ambas as consultas está associado à varredura da tabela *orders*, com filtro na coluna *orderdate*. Em ambos os planos esta junção é implementada por um operador *Nested Loop Join*. A execução da heurística seleciona, portanto, o particionamento horizontal na coluna *o_orderdate* da tabela *orders*. As faixas de valores são construídas a partir do histograma da coluna para cinco partições.

```
SELECT O_YEAR, SUM(CASE WHEN NATION = 'BRAZIL' THEN VOLUME ELSE 0
END)/SUM(VOLUME) AS MKT_SHARE
FROM (SELECT O_ORDERDATE AS O_YEAR, L_EXTENDEDPRICE*(1-L_DISCOUNT) AS
VOLUME, N2.N_NAME AS NATION FROM PART, SUPPLIER, LINEITEM, ORDERS, CUSTOMER,
NATION N1, NATION N2, REGION
WHERE P_PARTKEY = L_PARTKEY AND S_SUPPKEY = L_SUPPKEY
AND L_ORDERKEY = O_ORDERKEY AND O_CUSTKEY = C_CUSTKEY
AND C_NATIONKEY = N1.N_NATIONKEY
AND N1.N_REGIONKEY = R_REGIONKEY AND R_NAME = 'AMERICA'
AND S_NATIONKEY = N2.N_NATIONKEY
AND O_ORDERDATE BETWEEN '1995-01-01' AND '1996-12-31'
AND P_TYPE= 'ECONOMY ANODIZED STEEL') AS ALL_NATIONS
GROUP BY O_YEAR
ORDER BY O_YEAR
```

Figura 7: Consulta 8

A fragmentação horizontal derivada ocorre a partir da tabela particionada *orders*. Sua tabela-filha, *lineitem*, cuja chave primária é composta por (*l_orderkey*, *l_linenumber*), será particionada por referência a *orders*. Em *tpch-p* a execução da consulta 3 apresenta varreduras completas somente nas partições da tabela *orders* que satisfazem o predicado e nas partições correspondentes de *lineitem*. O plano de execução mostra *Nested Loop Join* para juntar as linhas das duas tabelas particionadas baseado no critério *orders.o_orderkey* = *lineitem.l_orderkey*. Nota-se que o otimizador percorreu somente as partições da tabela *lineitem* que correspondem às partições da tabela *orders*. A execução da consulta 8 em *tpch-p* também apresenta o mesmo comportamento (Figura 8). Para *orders*, somente as partições que satisfazem o predicado em *orderdate* são acessadas. Na figura 8 observa-se o benefício da estratégia.

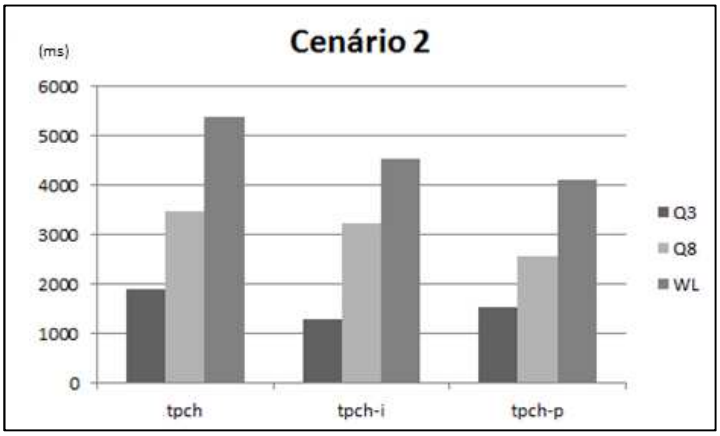

Figura 8: Cenário 2 – resultados

Qual o aumento do benefício do particionamento quando o volume de dados aumenta? Realizamos os mesmos testes em bases de dados com tamanhos 10GB e 50GB (Figura 9). Para Q8 o ganho de benefício relativo da estratégia de particionamento em relação à estratégia de índice foi de 21,67% no crescimento a 10GB e de 24,85% no crescimento a 50GB. Para a carga de trabalho (WL) o ganho de benefício relativo da estratégia de particionamento em relação à estratégia de índice foi de 16,58% no crescimento a 10GB e de 19,61% no crescimento a 50GB.

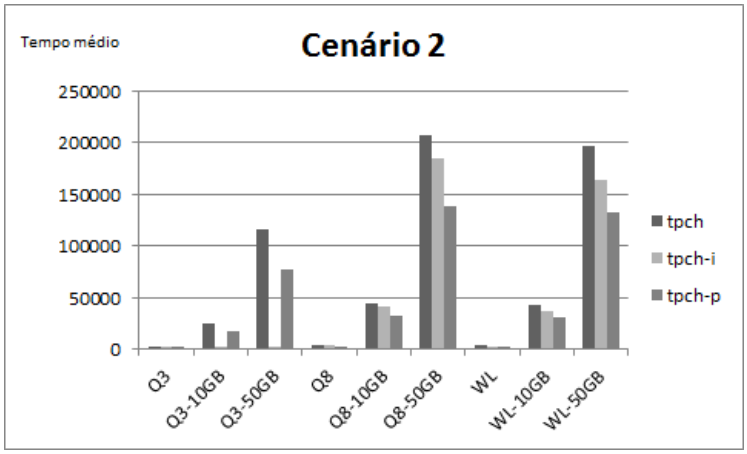

Figura 9: Cenário 2 - Aumento do volume de dados

4.3 – Cenário 3 - Atualizações

Seja a seguinte atualização executada no cenário 2, no qual a tabela *orders* foi particionada por *o_orderdate* e a tabela *lineitem* foi particionada por referência a *orders*.

```
UPDATE orders SET o_comment='Lorem ipsum dolor sit amet, '
WHERE o_orderdate BETWEEN '1994-06-01' AND '1997-10-01'
```

Esta atualização é executada em *tpch* através de um *Sequential Scan* em 29923,080 milisegundos. Em *tpch-p* esta mesma atualização é executada em 32237,294 milisegundos com *Sequential Scans* nas partições 2 a 5 de *orders*. Seja agora a atualização abaixo executada em outro cenário, no qual a tabela *lineitem* tenha sido particionada por *l_shipmode* (Cenário 3 – Figura 10).

```
UPDATE lineitem SET l_comment='Lorem ipsum dolor sit amet, '
WHERE l_shipmode IN ('MAIL', 'SHIP')
```

Esta atualização é executada em *tpch* através de um *Sequential Scan* em 150492,298 milisegundos. Em *tpch-p* esta mesma atualização é executada em 63459,862 milisegundos com *Sequential Scan* somente na partição correspondente de *lineitem*.

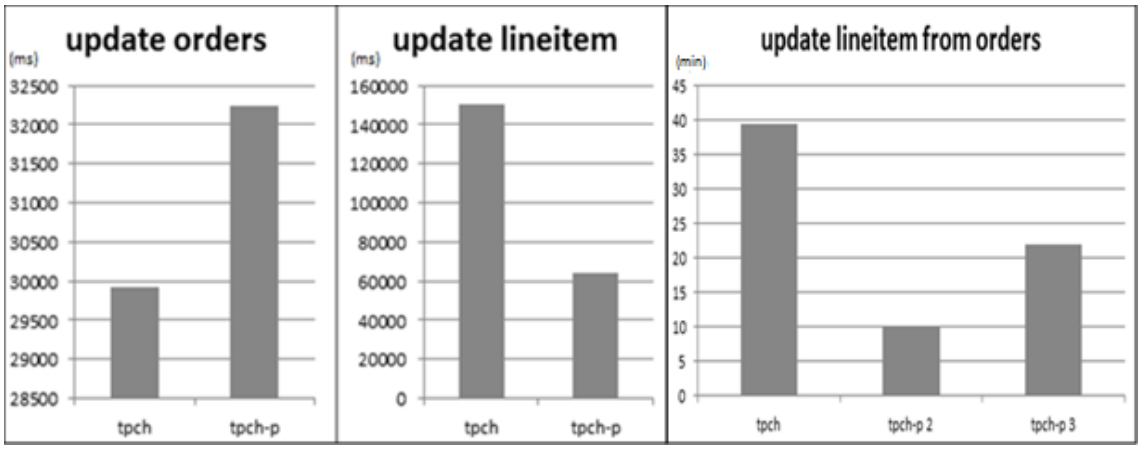

Figura 10: Cenário 3 – resultados

Os tempos do *update* no cenário 2 são semelhantes, porém os tempos medidos para a atualização no cenário 3 indicam que a atualização teve um desempenho melhor com a configuração de particionamento. Seja agora a atualização abaixo.

```
UPDATE lineitem AS l
SET l_comment='Lorem ipsum dolor sit amet, '
FROM orders AS o WHERE l.l_orderkey = o.o_orderkey
AND o_orderdate BETWEEN '1994-06-01' and '1997-10-01'
```

Esta atualização é executada em *tpch* em 39,33 minutos. Em *tpch-p* no cenário 2 esta mesma atualização é executada em 10,13 minutos e em *tpch-p* no cenário 3 esta mesma atualização é executada em 21,92 minutos.

5 – Conclusões e Trabalhos Futuros

Diante dos cenários estudados e dos resultados coletados, conclui-se que o particionamento horizontal é uma estratégia principal de sintonia fina, cujo ganho de desempenho pode ser comparado aos de índices, índices parciais e visões materializadas. Além disso, atualizações de dados também podem se beneficiar do particionamento.

Como trabalhos futuros estamos implementando as heurísticas na ferramenta de sintonia fina DBX (Almeida et. al, 2015), desenvolvida pelo grupo de pesquisa BioBD da PUC-Rio. Além disso, pretende-se (i) integrar o particionamento com as outras estratégias, como índices e visões materializadas, de forma que os benefícios das estratégias sejam estimados em conjunto, ao invés de o serem de forma independente, e (ii) estudar a evolução da heurística para bases de dados já particionadas, com a avaliação de possíveis cenários com alterações nos critérios de particionamento, incluindo a alteração na chave ou intervalo de particionamento utilizados, a criação de novos níveis de particionamento ou, simplesmente, a eliminação de partições.

Referências

Agrawal, S.; Narasayya, V. & Yang, B. (2004). Integrating Vertical and Horizontal Partitioning into Automated Physical Database Design. Anais da ACM International Conference on Management of Data (SIGMOD), pp. 359–370.

Almeida, A.C.B.; Brayner, A., Monteiro, J.M.; Lifschitz, S.; & Oliveira, R.P. (2015). DBX: um framework para auto-sintonia fina baseado em planos hipotéticos. Anais do SBBD Sessão de Demos, pp.149-154.

Bellatreche L., Schneider M., Lorinquer H. & Mohania M. (2004). Bringing Together Partitioning, Materialized Views and Indexes to Optimize Performance of Relational Data Warehouses, Anais da International Conference on Data Warehousing and Knowledge Discovery. DaWaK, pp. 15-25.

Costa, R. L. C.; Lifschitz, S.; Noronha, M. & Salles, M.V. (2005). Implementation of an agent architecture for automated index tuning. Anais da International Conference on Data Engineering (ICDE) Workshop on Self-managing Database Systems, pp. 1215.

Curino, C.; Jones, E.; Zhang, Y. & Madden, S. (2010). Schism: a workload-driven approach to database replication and partitioning. J. of PVLDB 3, I1-2, pp. 48–57.

Medeiros, A. S. (2017). Particionamento com Ação de Sintonia Fina. Dissertação de Mestrado, Departamento de Informática, PUC-Rio.

Monteiro, J. M. (2008). Uma abordagem não intrusiva para a manutenção automática do projeto físico de bancos de dados. Tese de Doutorado, Departamento de Informática, PUC-Rio.

TPC (2017). TPC-H Benchmark, http://www.tpc.org/tpch [Acessado em 24/05/2017].

Valduriez, P. & Özsu, M.T. (2011). Principles of Distributed Database Systems. 3rd. edition. Springer.

Wang, X.; Chen, J.; & Du, X. (2013). ASAWA: An Automatic Partition Key Selection Strategy. APWeb LNCS 7808, pp. 609–620.