Abordagem de suporte a transação através de consulta *HiveQL*

Juliano Gomes da Silveira, Tobias Stifft, Daiane Hemerich, Duncan Dubugras Alcoba Ruiz

Pontifícia Universidade Católica do Rio Grande do Sul (PUCRS)
Faculdade de Informática (FACIN)
Caixa Postal 14.19 – 90.619-900 – Porto Alegre – RS – Brazil

juliano.pro@gmail.com, stifft@gmail.com, daiane.hemerich@pucrs.br, duncan@pucrs.br

Abstract. *We propose in this paper an approach that simulates support for transactions on a Hadoop-Hive environment through a HiveQL query mechanism of temporal series. Such proposal has as premise to get round the limitation of Hadoop-Hive environment regarding UPDATE and DELETE operations, given that data persisted on Hadoop Distributed File System (HDFS) can be provided by a relational database.*

Resumo. *Propomos neste trabalho uma abordagem que simula o suporte a transações em um ambiente Hadoop-Hive através de um mecanismo de consulta HiveQL de série temporal. Tal proposta tem como premissa contornar a limitação do ambiente Hadoop-Hive quanto às operações de UPDATE e DELETE, uma vez que os dados persistidos no Hadoop Distributed File System (HDFS) podem ser providos por banco de dados relacional.*

1. Introdução

A plataforma *Apache Hadoop* [1] consolidou-se como uma sofisticada ferramenta para trabalhar com *Big Data*. Seu sistema de armazenamento distribuído (*Hadoop Distributed File System* - HDFS) [2][3] viabiliza performance no manejo de grandes conjuntos de dados em hardware de baixo custo. Além disso, o HDFS conta com recursos que o torna tolerante a falhas. O modelo de programação de *Hadoop* é inspirado em *MapReduce* [4], sendo este orientado para o processamento de grandes volumes de dados (estruturados ou não) sob uma arquitetura distribuída. Tal modelo divide o trabalho em pequenas tarefas independentes que são processadas em paralelo. Comumente, fazem uso de *Hadoop* sistemas de análise de logs, armazenamento e recuperação de dados oriundos de sistemas de sensores ou de identificação por radiofrequência (RFID), entre outras aplicações.

Dentre os diversos subprojetos associados à plataforma *Hadoop*, encontra-se o *Hive*, que foi inicialmente desenvolvido pelo *Facebook* e posteriormente admitido como um projeto de código aberto da *Apache*. O *Hive* provê um ambiente de abstração de

armazenamento e acesso a dados, focado em aplicações de *Data Warehouse*. Esta ferramenta é operada através de uma linguagem de alto nível, baseada em SQL ANSI (chamada *HiveQL*) e mantém abstrações equivalentes aos bancos de dados relacionais, tais como tabelas, atributos, registros, etc. *Hive* trabalha sobre os dados persistidos no HDFS e seus comandos *HiveQL* são convertidos em operações *MapReduce*, destacando-se as operações de sumarização, consulta e análise dos dados. Embora seja baseado em SQL, algumas extensões não são compatíveis com o *HiveQL*, como o suporte a transações e subconsultas.

Comumente, sistemas de banco de dados relacionais fornecem dados para o *Hadoop*, e, ainda que não seja um cenário típico de *Big Data*, existem aplicações que exigem suporte a transações, mesmo que seja apenas para exclusão e atualização de registros. Nesse ponto *Hadoop-Hive* é limitado, pois de forma nativa não admite operações de *DELETE* e *UPDATE* de registros [6], operações essas tomadas como triviais em ambientes de bancos de dados relacionais. Examinando a documentação, encontramos alguns subterfúgios para contornar esta limitação, como o uso de partições. Nesse caso, no momento do projeto, a tabela deve prever um particionamento de dados em seu nível elementar e, quando necessário, por uma operação de *INSERT OVERWRITE,* a partição é sobrescrita. Embora a atualização de partição funcione, esse recurso se torna paliativo, uma vez que o particionamento elementar gera fragmentação desnecessária. Para lidar com essa situação, neste trabalho propomos uma abordagem de suporte a transações para o *Hadoop-Hive*. Tal abordagem baseia-se em um mecanismo simplificado de armazenamento e consulta *HiveQL* que retorna os registros vigentes, ou seja, suprimindo aqueles que no SGDB de origem dos dados foram sobrescritos através de uma operação de *UPDATE* ou excluídos por uma operação de *DELETE*, sem a necessidade da utilização de artifícios adicionais, como o particionamento de tabela. Basicamente, a abordagem aqui demonstrada busca contornar a limitação de suporte transacional do *Hive* através de uma série temporal. Para tanto, primeiramente definimos algumas condições que os dados devem sustentar para que o mecanismo de suporte a transação possa operar. Após isso, detalhamos as operações da consulta *HiveQL*. Por fim, demonstramos alguns exemplos e experimentos. Ao longo do texto, para facilitar a compreensão, usaremos o acrônimo ASTC (abordagem para suporte a transação por consulta).

2. Abordagem de suporte a transação por consulta (ASTC)

Geralmente, *Hadoop* é integrado com SGDBs relacionais, tanto como receptor quanto fornecedor de dados. Para tanto, *Hive* conta com interfaces que viabilizam esse trabalho, como, por exemplo, os componentes de JDBC e ODBC. Porém, em virtude de seu foco, *Hadoop* possui algumas características nativas que contrastam com os SGDBs tradicionais [6]. Uma dessas é o suporte a transações, não admitindo atualizações e exclusões de registros.

Embora saibamos que este não é um cenário comum de aplicações de *Big Data*, a limitação da plataforma em processar atualizações e exclusões pode gerar alguns inconvenientes. Para ilustrar, consideremos o seguinte exemplo: uma tabela de SGDB que mantém registros de vendas de comércio eletrônico, onde cada registro representa uma compra, sendo que tal transação possui diversos status ('aguardando pagamento', 'despachado', 'entregue'). Supondo que esses dados sejam integrados com *Hadoop* e nessa integração esteja contemplado o código de venda (chave), valor, status e data de última atualização de status. Nesse caso, admitimos que para cada estímulo de *UPDATE* no SGDB o registro atualizado é encaminhado para o *Hadoop* ou posto em uma fila para posterior processamento em lote. Dada a forma em que esses dados foram concebidos e integrados, uma simples consulta para obter a quantidade de vendas por status pode ser inviável, mesmo usando uma ferramenta de consulta de alto nível como o *Hive*, tendo em vista que, no *Hadoop,* cada atualização incluirá uma nova entrada de dados, mantendo também as antigas entradas de registros de venda (com situação não vigente). Neste exemplo, se os dados fornecessem mais detalhes, como a data de pagamento, despacho e entrega, tornar-se-ia prático construir uma consulta para se analisar o panorama de vendas por status, uma vez que essas datas podem ser operadas pelo *HiveQL*. Porém, nem sempre se tem todos os dados necessários para o trabalho.

A ASTC pode ser usada como um artifício para lidar com a situação descrita acima. Esta abordagem pode ser resumida em três etapas: definição dos dados (onde são delineados alguns critérios aos quais os dados devem atender); redutos de integração (meio por onde os dados devem ser integrados do SGDB para o HDFS); e consulta (mecanismo de suporte a transação). As próximas subseções detalham as etapas.

2.1. Propriedade dos dados

Na ASTC, faz-se necessária a definição de duas propriedades que devem ser aplicadas sobre o SGDB, mais especificamente, pela tabela do SGDB que se deseja integrar com o *Hadoop*, conforme descrito a seguir:

i. A tabela do SGDB a ser integrada deve possuir uma chave candidata ou algum atributo que garanta a unicidade do registro;
ii. A tabela do SGDB a ser integrada deve possuir um atributo de série temporal contendo o apontamento da última modificação do registro. No caso de registros que nunca foram atualizados, tal atributo deve conter o apontamento do momento da inserção.

Na sequência deste trabalho, identificamos a chave candidata como atributo KEY e o atributo de série temporal como DAT.

2.2. Redutos de integração

A ASTC define o conceito de integração por dois redutos (reduto de *INSERT*/*UPDATE* e *DELETE*), sendo esses parte do processo de ETC (extração, transformação e carga de

dados). Esses redutos são o meio de transporte dos dados, podendo ser arquivos de texto, tabelas persistidas no próprio SGDB a ser integrado, ou qualquer outro tipo compatível com *Hadoop*. No caso de arquivo texto, uma opção é utilizar um mecanismo de gatilho externo, para que a cada estímulo de *INSERT*, *UPDATE* e *DELETE* executado no SGDB, uma nova entrada seja inserida no arquivo, sendo na sequência esses arquivos integrados ao HDFS. Se os redutos forem concebidos no próprio SGDB através de tabelas, uma opção é utilizar um gatilho interno, onde cada estímulo gera uma nova entrada para a respectiva tabela e na sequência os dados são integrados ao HDFS via driver ODBC/JDBC, disponível no *Hive*. Neste trabalho usamos a opção de integração via tabela, ou seja, cada reduto é uma tabela persistida no SGDB com o mesmo layout da tabela a ser integrada, onde cada estímulo de *INSERT*, *UPDATE* e *DELETE* é capturado por um gatilho e o registro é inserido no respectivo reduto. Necessita-se, portanto, construir três gatilhos, um para cada comando. O ponto determinante aqui é que a cada operação o registro seja encaminhado para o respectivo reduto, ou seja, uma operação de *INSERT* ou *UPDATE* sobre a tabela a ser integrada deve gerar uma entrada no reduto de *INSERT*/*UPDATE*. O mesmo vale para as operações de *DELETE*, onde as entradas devem ser escritas sobre o reduto de *DELETE*. A fim de facilitar a compreensão, vamos nomear os redutos como: RED_INSERT_UPDATE e RED_DELETE. Ao final, os redutos são movidos para HDFS. Neste trabalho, para transporte dos dados utilizamos o componente de ODBC disponível no *Hive*. A Figura 1 ilustra o mecanismo de redutos adotado neste trabalho.

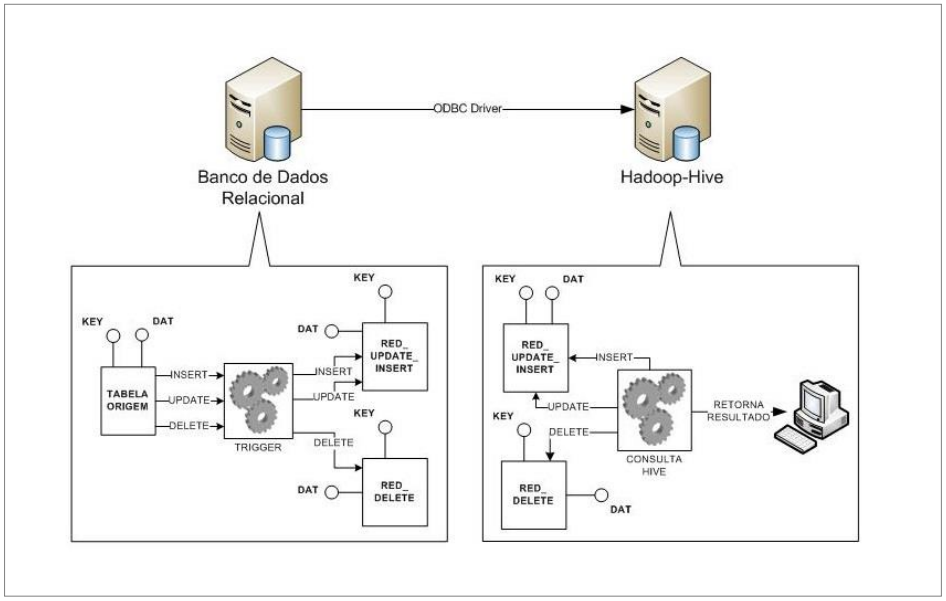

Figura 1. Mecanismo de redutos de integração utilizado neste trabalho.

2.3. Consulta *HiveQL*

A consulta foi elaborada com objetivo de abstrair os registros não vigentes, ou seja, aqueles que foram excluídos ou que foram sobrescritos por atualização. Tal consulta foi construída sobre o *Hive* utilizando apenas operações comuns, como junções, uniões e

operadores lógicos sobre o par key/dat. No Quadro 1 é demonstrada a consulta *HiveQL*. Cada fragmento numerado do código é detalhado na sequência do texto.

```
select red_insert_update.*
  from red_insert_update
  join (select mx_ins_upd.key, mx_ins_upd.dat
          from (select red_insert_update.key, max(red_insert_update.dat) dat
                  from red_insert_update
                 group by red_insert_update.key) mx_ins_upd                    2.3.1
          left outer join (select red_delete.key, max(red_delete.dat) dat
                             from red_delete
                            group by red_delete.key) mx_del                    2.3.2
            on (mx_ins_upd.key = mx_del.key)                                   2.3.3
         where mx_del.key is null
            or mx_ins_upd.dat > mx_del.dat) mx                                 2.3.4   2.3.5
    on (mx.key = red_insert_update.key)
 where mx.dat = red_insert_update.dat
```

Quadro 1. Consulta *Hive-QL*.

2.3.1. Neste trecho RED_INSERT_UPDATE é consultada abstraindo aqueles registros que foram sobrescritos através da operação de *UPDATE*. Para tanto, se utiliza a função MAX sobre série temporal (DAT), agrupando pela chave (KEY). Com isso, tem-se a situação mais atual de cada chave;

2.3.2. Neste ponto são consultados os registros persistidos em RED_DELETE, ou seja, aqueles que foram excluídos. Porém, há situações em que, segundo a chave, um mesmo registro é excluído mais de uma vez, portanto nesse trecho aplica-se a função MAX sobre a série temporal (DAT), agrupando pela chave (KEY). Desta forma tem-se como resultado o par de atributos (DAT/KEY) da última exclusão;

2.3.3. Esta junção estabelece a relação entre o trecho 2.3.1 (mx_ins_upd) e o trecho 2.3.2 (de mx_del) através da chave (KEY). Neste caso, utiliza-se a junção *left outer join*, pois tal consulta deve retornar todos os registros de mx_ins_upd independentemente se há um respectivo representante em mx_del;

2.3.4. Neste trecho aplica-se a cláusula *WHERE* sobre os atributos mx_del.key, mx_del.dat e mx_ins_upd.dat. Aqui se iguala a nulo o atributo mx_del.key, para eliminar os registros que foram excluídos. Há situações em que um registro excluído pode ser posteriormente reinserido. A cláusula seguinte (mx_ins_upd.dat > mx_del.dat) evita que esses registros sejam desconsiderados;

2.3.5. Este bloco resulta em pares de valores (KEY, DAT) dos registros vigentes.

Na próxima seção, cada um destes trechos de código é executado e comentado sobre uma amostra de dados de exemplo.

3. Exemplo

Para melhor compreender a ação do mecanismo de suporte a transações por consulta, nessa seção vamos usar um exemplo prático, no qual simulamos a ação da consulta *HiveQL* (seção 2.2) sobre as estruturas RED_INSERT_UPDATE e RED_DELETE. Abaixo cada trecho da consulta é executado, comentado e exposto o resultado.

As tabelas abaixo representam os redutos RED_INSERT_UPDATE e RED_DELETE, respectivamente.

Tabela 1. Reduto RED_INSERT_UPDATE

KEY	DAT	AUTHOR	BOOK
1	01/08/2013	KNUTH; DONALD E.	ART OF COMPUTER PROGRAMMING V.1
2	01/08/2013	TANENBAUM; ANDREW S.	OPERATING SYSTEMS DESIGN AND IMPLEMENTATION
3	01/08/2013	STEWART; JAMES	ESSENTIAL CALCULUS
4	01/08/2013	HAN; JIAWEI	DATA MINING CONCEPTS AND TECHNIQUES
1	02/08/2013	KNUTH; DONALD ERVIN	ART OF COMPUTER PROGRAMMING V.1
2	03/08/2013	TANENBAUM; ANDREW S.	OPERATING SYSTEMS DESIGN AND IMPLEMENTATION
2	04/08/2013	TANENBAUM; ANDREW STUART	OPERATING SYSTEMS DESIGN AND IMPLEMENTATION
2	06/08/2013	TANENBAUM; ANDREW STUART	OPERATING SYSTEMS DESIGN AND IMPLEMENTATION

Tabela 2. Reduto RED_DELETE

KEY	DAT	AUTHOR	BOOK
2	02/08/2013	TANENBAUM; ANDREW S.	OPERATING SYSTEMS DESIGN AND IMPLEMENTATION
4	02/08/2013	HAN; JIAWEI	DATA MINING CONCEPTS AND TECHNIQUES
2	05/08/2013	TANENBAUM; ANDREW STUART	OPERATING SYSTEMS DESIGN AND IMPLEMENTATION

Observando os dados de cada reduto, podemos desdobrar a série temporal das transações que ocorreram na tabela do SGDB, conforme abaixo:

i. Em 01/08/2013 são inseridos os registros identificados como KEY(1, 2, 3, 4);

ii. Em 02/08/2013 é atualizado o atributo *AUTHOR* do registro KEY(1);

iii. Em 02/08/2013 os registros KEY(2, 4) são excluídos;

iv. Em 03/08/2013 volta a ser inserido o registro identificado como KEY(2);

v. Em 04/08/2013 o registro KEY(2) sofre uma atualização no atributo *AUTHOR*.

vi. Em 05/08/2013 o registro KEY(2) é excluído novamente;

vii. Em 05/08/2013 volta a ser inserido o registro identificado como KEY(2).

Quando executado o trecho 2.2.1 da consulta, temos o resultado de pares (KEY/DAT) listado na Tabela 3, onde constam os registros que estão persistidos em RED_INSERT_UPDATE desconsiderando aqueles desatualizados, ou seja, aqueles que segundo o par KEY/DAT não estão vigentes.

Tabela 3. Resultado parcial - consulta 2.2.1

KEY	DAT
3	01/08/2013
4	01/08/2013
1	02/08/2013
2	06/08/2013

Ao executar o trecho 2.2.2 da consulta, tem-se o resultado de pares (KEY/DAT) listado na Tabela 4. Da mesma forma que a consulta 2.2.1, também busca o registro mais atual da série temporal para cada chave, através da função MAX no atributo DAT.

Tabela 4. Resultado parcial - consulta 2.2.2

KEY	DAT
4	02/08/2013
2	05/08/2013

O trecho 2.2.5 da consulta utiliza-se do resultado do trecho 2.2.1 e 2.2.2 para encontrar os pares (KEY/DAT) vigentes. Neste caso, através da junção 2.3.3 e da cláusula 2.2.4 além de buscar os registros mais atuais, faz a diferenciação de exclusões que foram novamente inseridas. A Tabela 5 exibe o resultado do par (KEY/DAT).

Tabela 5. Resultado parcial - consulta 2.2.5

KEY	DAT
3	01/08/2013
1	02/08/2013
2	06/08/2013

Por fim, a consulta completa retorna todos os dados, restringindo os valores através dos pares de valores resultantes do trecho 2.3.5, conforme demonstrado abaixo.

Tabela 6. Resultado da consulta completa

KEY	DAT	AUTHOR	BOOK
3	01/08/2013	STEWART; JAMES	ESSENTIAL CALCULUS
1	02/08/2013	KNUTH; DONALD ERVIN	ART OF COMPUTER PROGRAMMING V.1
2	06/08/2013	TANENBAUM; ANDREW STUART	OPERATING SYSTEMS DESIGN AND IMPLEMENTATION

Na tabela acima verificamos que o registro representado por KEY(4) não foi exibido, pois, segundo RED_DELETE, este registro foi excluído na sequência e não foi inserido novamente. No caso de KEY(2), que também foi excluído, a consulta o preservou no resultado, pois segundo a série temporal houve uma nova inserção após a exclusão. O registro representado por KEY(1) foi retornado pela consulta, porém se observa que haviam dois registros para essa chave, e neste caso a consulta considerou apenas o registro mais atual (DAT = 02/08/2013). O registro identificado por KEY(3) também permaneceu, pois neste não houve atualizações ou exclusões.

4. Testes

A fim de testar o desempenho da abordagem aqui proposta foram realizados alguns experimentos, nos quais foi avaliado o tempo de resposta da ASTC. Para tanto, foram processados 10 conjuntos de dados de diferentes tamanhos, sendo o menor com 1.650.000 registros, crescendo linearmente até o maior, de 16.500.000. Os conjuntos continham registros de inserção, atualização, exclusão e portanto mantiveram-se segregados nos redutos RED_INSERT_UPDATE e RED_DELETE. O reduto RED_INSERT_UPDATE continha todos os registros que foram em algum momento inseridos ou atualizados na tabela de banco de dados origem, ou seja, registros capturados pelos gatilhos de *INSERT*/*UPDATE*. Da mesma forma, o reduto

RED_DELETE continha todos os registros que foram em algum momento excluídos da tabela de banco de dados origem, ou seja, registros capturados pelo gatilho de *DELETE*.

Os experimentos aqui demonstrados foram executados sobre 4 nós virtuais. O sistema operacional utilizado foi *Linux Red Hat* 5.5, sendo cada um dos nós virtuais de 2GB de memória RAM, compartilhando o processador Intel i5-2450M de 4 núcleos de 2.5 GHz. As versões utilizadas de *Hadoop* e *Hive* são 1.1.1 e 0.9.0, respectivamente.

4.1. Conjunto de dados

Todos os conjuntos de dados testados possuíam inserções, atualizações, exclusões e inserções pós-exclusão. Chama-se de registros de inserção pós-exclusão aqueles que segundo o par KEY/DAT foi inserido, excluído e na sequência voltou a ser inserido. O gráfico abaixo demostra a composição de registros dos 10 conjuntos utilizados.

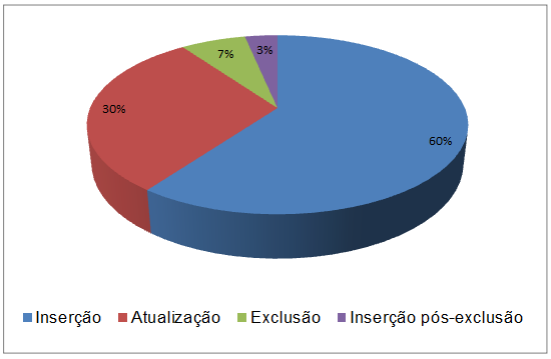

Figura 2. Composição dos registros de teste.

4.2. Consultas de teste

Para cada um dos 10 conjuntos, a consulta utilizando ASTC foi executada, incluindo-se em seu escopo a função *COUNT*(*). Adicionalmente, para fins de comparação, executamos consultas de contagem sobre os mesmos conjuntos sem o uso da ASTC. A intenção dos testes foi verificar o desempenho da abordagem ASTC para uma consulta de contagem dos registros vigentes frente a execução de uma consulta simples, que considera todos os registros (vigentes e não vigentes). Nos quadros seguintes são demonstradas ambas as consultas.

```
select count(*)
  from red_insert_update
  join (select mx_ins_upd.key, mx_ins_upd.dat
          from (select red_insert_update.key, max(red_insert_update.dat) dat
                  from red_insert_update
                group by red_insert_update.key) mx_ins_upd
          left outer join (select red_delete.key, max(red_delete.dat) dat
                             from red_delete
                           group by red_delete.key) mx_del
            on (mx_ins_upd.key = mx_del.key)
         where mx_del.key is null
            or mx_ins_upd.dat > mx_del.dat) mx
    on (mx.key = red_insert_update.key)
 where mx.dat = red_insert_update.dat
```

Quadro 3. Consulta ASTC.

```
select count(*)
  from red_insert_update
```

Quadro 4. Consulta simples.

4.3. Análise dos resultados

Os tempos de consulta de ambas as consultas foram plotados em um gráfico de linhas, para facilitar a analise dos resultados.

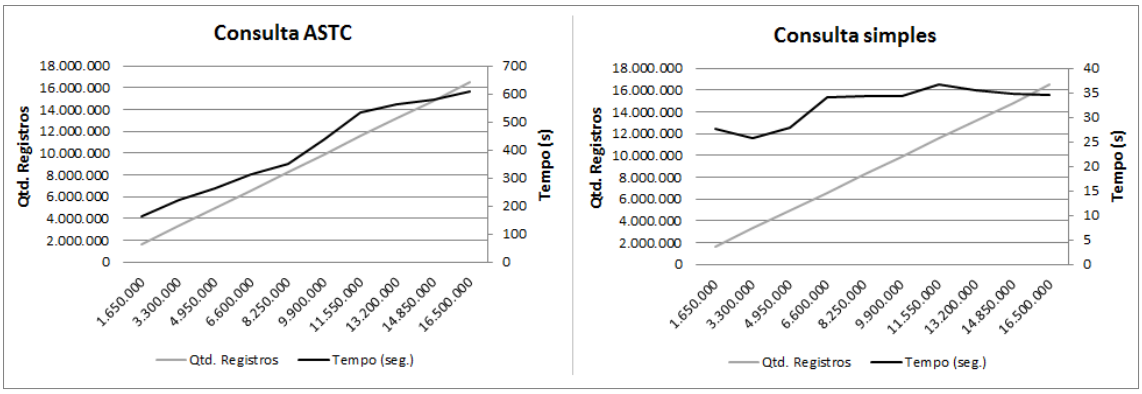

Figura 3. Comparação dos tempos de consulta.

Os gráficos acima possuem a seguinte composição: as escalas do eixo x e do eixo y (esquerdo) representam a quantidade total de registros processados; o eixo y direito representa a escalada de tempo de execução; a linha cinza exibe o crescimento da quantidade de registros processados; a linha preta demonstra a curva de tempo das consultas. Ao observar os indicadores exibidos na Figura 3, constatamos que o tempo de resposta da consulta que usa ASTC foi significativamente maior em relação ao tempo de resposta da consulta que não usou ASTC. Entendemos que esse comportamento deu-se em função das sincronizações realizadas pelas relações de junção e a aplicação da função *MAX* sobre o atributo DAT. O indicador de tempo da consulta ASTC permaneceu associado ao crescimento dos registros, com uma leve queda à medida em que a quantidade de registros se eleva. Porém, ao observar o mesmo indicador sem o uso de ASTC, identificamos que a relação entre o crescimento dos dados e o tempo de execução é menos aparente. Logo, considerando o ambiente utilizado neste experimento e os dados processados nos testes, podemos concluir que o tempo de resposta da consulta da ASTC tem uma forte relação com o volume de dados processados.

5. Considerações Finais

A abordagem proposta neste trabalho mostrou-se efetiva na tarefa de simular o comportamento de suporte a transações. Pode ser utilizada em um ambiente *Hadoop-Hive,* onde se deseja obter uma visão das transações vigentes, desconsiderando aquelas transações que foram invalidadas por operações de *UPDATE* ou *DELETE* na origem.

Ao observar os resultados apresentados na subseção 4.3, percebemos a clara queda de desempenho (tempo de resposta) quando empregada a consulta ASTC frente à consulta de contagem simples. Tal queda já era prevista, pois a complexidade da consulta ASTC é significativamente maior em função das suas junções e o uso da operação *MAX*. Porém o tempo de resposta de ASTC permaneceu associado ao crescimento dos dados, sendo que no mesmo indicador da consulta simples essa associação se manteve fraca, ou seja, a complexidade da consulta ASTC se manteve linear ao crescimento dos dados, diferentemente da consulta simples onde a linha nos indica uma escala mais logarítmica do que linear (Figura 3). Esse resultado nos convida a progredir essa pesquisa, investigando as causas deste crescimento linear.

Referências

[1] The Apache Software Foundation, Apache-Hadoop, 2013. Disponível em: http://hadoop.apache.org. Acesso em: agosto de 2013.

[2] Shvachko, K., Kuang, H., Radia, S., Chansler, R., The Hadoop Distributed File System. In: Proceedings of the 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST), 2010, pp. 1-10.

[3] Ghemawat, S., Gobioff, H., Leung, S., The Google File System. In: Proceedings of the 19th ACM Symposium on Operating Systems Principles, 2003, pp. 20-43.

[4] Dean, J., Ghemawat, S., MapReduce: Simplified Data Processing on Large Clusters. In: Proceedings of the 6th conference on Symposium on Opearting Systems Design & Implementation (OSDI), 2004, pp. 1-10.

[5] The Apache Software Foundation, Hadoop-Hive, 2013. Disponível em: http://hive.apache.org/. Acessado em: Agosto de 2013.

[6] The Apache Software Foundation, Apache Wiki (Hive Tutorial). Disponível em: https://cwiki.apache.org/confluence/display/Hive/Tutorial. Acessado em: Agosto de 2013.

[7] Jia, B., WiktorWlodarczyk, T., Rong, C., Performance Considerations of Data Acquisition in Hadoop System. In: 2nd IEEE International Conference on Cloud Computing Technology and Science, 2010, pp. 545-549.

[8] Goldman, A., Kon, F., Junior, F. P., Polato, I., Pereira, R. P., Apache Hadoop: Conceitos teóricos e práticos, evolução e novas possibilidades. In: XXXI Jornadas de atualizações em informática, 2012, pp. 88-136.