Análise Experimental de Bases de Dados Relacionais e NoSQL no Processamento de Consultas sobre Data Warehouse

Conference Paper · November 2012

CITATIONS
2

READS
693

6 authors, including:

Anderson Chaves Carniel
Federal University of São Carlos
48 PUBLICATIONS **261** CITATIONS

SEE PROFILE

Marcela Xavier Ribeiro
Federal University of São Carlos
65 PUBLICATIONS **568** CITATIONS

SEE PROFILE

Ricardo Rodrigues Ciferri
Federal University of São Carlos (UFSCar)
93 PUBLICATIONS **761** CITATIONS

SEE PROFILE

Análise Experimental de Bases de Dados Relacionais e NoSQL no Processamento de Consultas sobre Data Warehouse

Anderson Chaves Carniel[1], Aried de Aguiar Sá[1], Marcela Xavier Ribeiro[1], Renato Bueno, Cristina Dutra de Aguiar Ciferri[2], Ricardo Rodrigues Ciferri[1]

[1] Universidade Federal de São Carlos
[2] Universidade de São Paulo
accarniel@gmail.com, aried.sa@dc.ufscar.br, renato@dc.ufscar.br, cdac@icmc.usp.br, ricardo@dc.ufscar.br

Abstract. Data warehouse (DW) is a large, oriented-subject, non-volatile, and historical database, and an important component of Business Intelligence. On DW are executed OLAP (Online Analytical Processing) queries that often culminate in a high response time. Fragmentation of data, materialized views and indices aim to improve performance in processing these queries. Additionally, NoSQL (Not only SQL) database are used instead of the relational database, to improve specific aspects such as performance in query processing. In this sense, in this paper is investigated and compared DW implementations using relational databases and NoSQL. We evaluated the response times in processing queries, memory usage and CPU usage percentage, considering the queries of the Star Schema Benchmark. As a result, the column-oriented model implemented by the software FastBit, showed gains in time of 25.4% to 99.8% when compared to other NoSQL models and relational in query processing.

Resumo. *Data warehouse* (DW) é uma base de dados orientada à assunto, não volátil, histórica e volumosa, sendo um componente importante da inteligência de negócio. Sobre DW incidem consultas OLAP (*Online Analytical Processing*) que frequentemente culminam em um alto tempo de resposta. Fragmentação de dados, visões materializadas e estruturas de indexação objetivam melhorar o desempenho no processamento dessas consultas. Adicionalmente, banco de dados NoSQL (*Not only SQL*) são usados como alternativa dos banco de dados relacionais, visando melhorar aspectos específicos, tal como o desempenho no processamento de consultas. Nesse sentido, neste trabalho é investigado e comparado implementações de DW usando banco de dados relacionais e NoSQL. Foram avaliados os tempos de respostas no processamento de consultas, o uso de memória e o uso porcentual de CPU, considerando as consultas do *Star Schema Benchmark*. Como resultado, o modelo orientado a coluna implementado pelo *software* FastBit, apresentou ganhos de reduções de tempo de 25,4% a 99,8% se comparado aos outros modelos NoSQL e relacional, no processamento de consultas.

Categories and Subject Descriptors: H. Information Systems [**H.m. Miscellaneous**]: Databases

Keywords: data warehouse, query processing, experimental evaluation, relational databases, NoSQL, bitmap join indices

1. INTRODUÇÃO

Um *data warehouse* (DW) integra informações de diversas fontes para auxiliar na tomada de decisão estratégica, compreendendo em uma base de dados histórica, orientada a assunto e não volátil [Kimball and Ross 2002]. Como um dos componentes principais da inteligência de negócio, sobre um DW são executadas consultas analíticas comumente processadas por ferramentas OLAP (*Online Analytical Processing*) [Xu et al. 2007]. Por envolver um grande volume de dados, tais consultas exigem um grande esforço computacional, tornando um desafio processá-las eficientemente.

Técnicas como a fragmentação dos dados [Golfarelli et al. 2000], visões materializadas [Baikousi and Vassiliadis 2009] e estruturas de indexação [O'Neil and Graefe 1995; Stockinger and Wu 2006], visam melhorar o tempo de resposta das consultas OLAP. Além disso, NoSQL tem sido cada vez mais

usado para processar grandes volumes de dados, sendo aplicáveis em situações com características próprias, tal como o processamento de consultas em dados não voláteis [Bonnet 2011; Cattel 2010; Han 2011]. Tendo em vista estas variadas formas, neste artigo é investigado e comparado o uso de fragmentação de dados, visões materializadas e estruturas de indexação com os modelos NoSQL para processar consultas OLAP sobre DW. Nesse contexto, as contribuições deste artigo são análises realizadas a partir de comparações experimentais executadas por meio de testes de desempenho considerando os modelos orientado a colunas e a documentos do NoSQL com o modelo relacional. Com isso, identificam-se mecanismos de consulta eficientes para o processamento de consultas sobre DW, auxiliando na elaboração de ferramentas OLAP para processar consultas eficientemente, tal como em [Carniel and Siqueira 2011].

Este artigo está organizado da seguinte forma. Na seção 2 é apresentada a fundamentação teórica, mostrando os conceitos necessários para a compreensão deste trabalho. Na seção 3 são discutidos e apresentados os resultados de testes experimentais, que avaliaram o tempo de resposta de consultas, consumo de memória e porcentual de CPU. Finalmente, na seção 4 concluem-se os resultados obtidos e o direcionamento para trabalhos futuros.

2. FUNDAMENTAÇÃO TEÓRICA

2.1 Data warehouse e o processamento de consultas

DW é uma base de dados histórica, integrada, não volátil e orientada ao assunto, utilizada para o armazenamento de dados visando auxiliar na tomada de decisão estratégica [Kimball and Ross 2002; Xu 2007]. Um DW pode ser implementado usando o esquema estrela ou floco de neve por meio do modelo relacional. Tabelas de fatos e de dimensão compõem tais esquemas [Kimball and Ross 2002]. Uma tabela de fatos armazena as medidas quantitativas do negócio analisado, enquanto uma tabela de dimensão caracteriza o assunto e seus atributos podem formar hierarquias. Na Fig. 1 é mostrado um esquema estrela composto pela tabela de fatos *Lineorder* e tabelas de dimensão *Customer*, *Supplier*, *Date* e *Part* [O'Neil et al. 2009]. Os atributos $s\_region \leq s\_nation \leq s\_city \leq s\_address$ da tabela de dimensão *Supplier* constituem uma hierarquia permitindo a agregação dos dados e o processamento de consultas *drill-down* e *roll-up*, comumente usadas em aplicações OLAP. Um esquema floco de neve difere do esquema estrela por normalizar as hierarquias envolvidas, aumentando consequentemente o número de junções envolvidas.

No processamento de consultas, a técnica de junção estrela se torna menos eficiente por envolver junções, agrupamentos, filtros e ordenações sobre um grande volume de dados, culminando um grande tempo de resposta. Na literatura existem técnicas para melhorar o desempenho no processamento de consultas, tais como a (i) fragmentação dos dados [Golfarelli et al. 2000], o uso de (ii) visões materializadas [Baikousi and Vassiliadis 2009] e (iii) estruturas de indexação [O'Neil and Graefe 1995; Stockinger and Wu 2006]. Na técnica de fragmentação dos dados são construídas visões fragmentadas verticalmente (VFV) que objetivam eliminar junções entre as tabelas de dimensão e de fatos, mantendo um conjunto mínimo de atributos necessários para responder um conjunto de consultas [Golfarelli et al. 2000]. Por exemplo, pode-se construir uma VFV sobre o esquema estrela da Fig. 1, usando a álgebra relacional: $\Pi_{d\_year,\ p\_category,\ p\_brand1,\ s\_region,\ lo\_revenue}$ (*Part* ⋈ *Lineorder* ⋈ *Supplier* ⋈ *Date*). Portanto, ao se usar uma VFV para processar uma consulta, apenas os filtros, ordenações e agrupamentos serão computados. Por outro lado, uma visão materializada (VM) além de eliminar as junções, armazena os dados previamente agrupados e os resultados das funções de agregações sobre as medidas [Baikousi and Vassiliadis 2009], assim diminuindo o volume de dados se comparado a uma VFV. Por exemplo, uma visão materializada sobre o esquema estrela da Fig. 1 pode ser construída usando a álgebra relacional: $_{d\_year,\ p\_category,\ p\_brand1,\ s\_region,}\ G_{SUM(lo\_revenue)}\ (\Pi_{d\_year,\ p\_category,\ p\_brand1,\ s\_region,\ lo\_revenue}$ (*Part* ⋈ *Lineorder* ⋈ *Supplier* ⋈ *Date*)). Portanto, ao se usar uma VM para processar uma consulta, apenas os filtros e ordenações serão computados. Estrutura de indexação é

outro fator importante para melhorar o desempenho no processamento das consultas. O índice *bitmap* de junção [O'Neil and Graefe 1995] é comumente utilizado para processar consultas sobre DW, tendo como principal vantagem a eliminação de junção entre as tabelas de fatos e de dimensão. Um índice *bitmap* de junção é criado sobre atributos das tabelas de dimensão, construindo-se vetores de *bits* para cada valor distinto dos atributos [O'Neil and Graefe 2995; Stockinger and Wu 2006]. O *i*-ésimo *bit* do vetor de *bits* armazenará o valor 1 se o valor correspondente ocorre na *i*-ésima tupla da tabela de fatos. Caso contrário, o *bit* será 0. Por construir um vetor de *bits* para cada valor distinto, a cardinalidade de atributos é um fator que pode diminuir a eficiência deste índice. Para contornar este problema, técnicas de encaixotamento, codificação e compressão são utilizadas [Stockinger and Wu 2006].

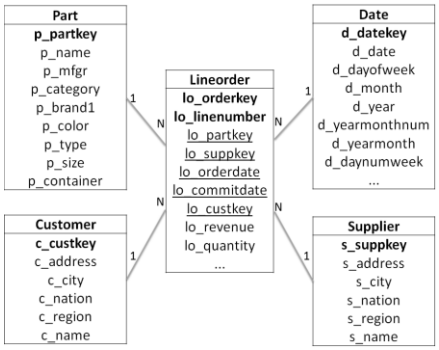


Fig. 1. Esquema estrela de uma aplicação a varejo [O'Neil et al. 2009]

2.2 Banco de Dados NoSQL

Uma característica interessante dos bancos de dados NoSQL (*Not only SQL*) para o contexto de DW é o armazenamento e processamento de consultas envolvendo grandes volumes de dados [Cattel 2010]. Banco de dados NoSQL baseiam-se no paradigma BASE (*Basically Available, Soft State, Eventually consistent*) ao contrário do ACID (*Atomic, Consistent, Isolation and Durable*) dos banco de dados relacionais [Bonnet 2011]. Além disso, sacrificando um dos três aspectos do teorema CAP (*Consistent, Available, Partition*) busca-se a maximização dos outros dois [Bonnet 2011; Cattel 2010]. Por exemplo, um banco de dados NoSQL pode maximizar a consistência e disponibilidade sacrificando o particionamento dos dados. Os principais modelos das bases de dados NoSQL são: (i) chave-valor [Han 2011]; (ii) orientado a coluna [Cattel 2010; Liu et al. 2011] e (iii) orientado a documentos [Bonnet 2011].

No modelo chave-valor os dados são estruturados como pares de chaves e valores, comumente expressos por índices *hash* [Han 2011]. Todas as operações de manipulação e consulta dos dados são realizadas sobre as chaves, com um alto desempenho. Consultas *ad-hoc* nesse modelo de dados é inviável, uma vez que todas as consultas são sobre as chaves e não sobre os valores. Já no modelo orientado a coluna, os dados são armazenados de acordo com suas colunas independentemente [Liu 2011]. Cada coluna é exclusivamente armazenada em cada tabela, sem a existência de relacionamentos. Finalmente, o modelo orientado a documento estende o modelo chave-valor, associando a uma chave um documento, por exemplo, documentos JSON (*JavaScript Object Notation*) [Bonnet 2011]. Em adição, no modelo orientado a documento é possível a execução de consultas pelos valores contidos nos documentos, bem como sua indexação, para o processamento de consultas *ad-hoc* [Bonnet 2011; Han 2011].

Neste artigo foram avaliados os modelos orientados a coluna e documento, capazes de processar consultas *ad-hoc*. Para o modelo orientado a coluna foram selecionados os *softwares* FastBit [Rübel et

al. 2009] e LucidDB [Liu 2011, LucidDB]. Eles utilizam índices *bitmap* como estrutura de indexação, além de algoritmos de compressão. Adicionalmente, o LucidDB foi criado para o ambiente de *data warehousing*, justificando a sua escolha. Além disso, o mesmo armazena estatísticas e algoritmos de organização dos vetores de *bits* em árvores-B para o processamento de consultas. A escolha do *software* FastBit foi motivada pelos resultados mostrados ao se processar consultas OLAP [Carniel and Siqueira 2011]. Já para o modelo orientado a documento foi selecionado o MongoDB, que também armazena estatísticas para melhorar o desempenho no processamento de consultas. Todos os *softwares* avaliados são livres e os resultados dos testes experimentais são discutidos na seção 3.

3. COMPARAÇÃO DE DESEMPENHO

Nesta seção são discutidos e apresentados os resultados dos testes de desempenho experimentais realizados considerando duas bases de dados denominadas DW1 e DW10. Tais bases de dados são idênticas ao esquema estrela da Fig. 1 e foram geradas sinteticamente por meio do *Star Schema Benchmark* (SSB) [O'Neil et al. 2009]. Enquanto a base de dados DW1 foi construída com o fator de escala 1 contendo 6 milhões de tuplas na tabela de fatos, a base de dados DW10 é 10 vezes mais volumosa, construída com o fator de escala 10 contendo 60 milhões de tuplas na tabela de fatos. Foi investigado o desempenho dos *softwares* FastBit, LucidDB, MongoDB baseados em NoSQL, além do sistema gerenciador de banco de dados (SGBD) relacional PostgreSQL. No *software* FastBit foram construídos índices *bitmap* de junção para cada VFV e VM com o algoritmo de compressão WAH [Stockinger and Wu 2006], com codificação e sem *binning*. Enquanto no LucidDB, além de manter as bases de dados DW1 e DW10, foram construídos índices *bitmap* de junção com algoritmo de compressão para cada VFV e VM. No MongoDB existiu a necessidade de mapear o esquema estrela da Fig. 1 para o armazenamento em documentos. Dessa forma, a tabela de fatos do esquema estrela foi representada por uma coleção (*Lineorder*), tendo como documentos incorporados suas dimensões (*Customer, Part, Supplier* e *Date*). Além disso, outras coleções foram criadas no MongoDB para o armazenamento de VFV e VM. Uma VFV no MongoDB não contém documentos incorporados, enquanto uma VM além de não conter documentos incorporados, os dados estão previamente agrupados e suas medidas calculadas. Por fim, no SGBD PostgreSQL, além das bases de dados DW1 e DW10, construiu-se uma VFV e VM correspondente para cada consulta.

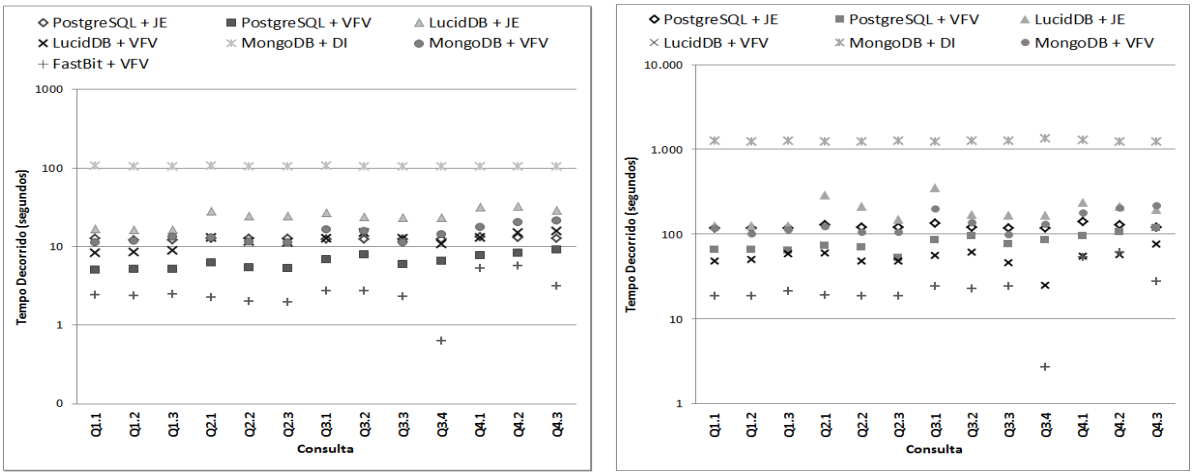
As 13 consultas do SSB, divididas em quatro grupos de consultas com complexidade crescente (Q1, Q2, Q3 e Q4), foram executadas na plataforma de *hardware* e *software* que segue. Um computador com um processador Intel(R) Pentium(R) D com frequência de 2,80Ghz, disco rígido SATA de 320 GB com 7200 RPM, e 2 GB de memória principal. O sistema operacional foi Fedora 16 com a versão do Kernel 3.1.0-7.fc16.x86_64, com os *softwares*: FastBit 1.3.0, LucidDB 0.9.4, MongoDB 2.0.5-rc1, PostgreSQL 9.1.3 e Java JDK 1.7.0_04. Na seção 3.1 são apresentados os resultados relativos ao tempo de execução das consultas utilizando a técnica de VFV e VM sobre as bases de dados DW1 e DW10 para cada ferramenta avaliada. Por fim, na seção 3.2 é mostrado o uso de memória e de CPU.

3.1 Processamento de Consultas Usando Visões Fragmentadas Verticalmente e Visões Materializadas

Inicialmente, foram executadas as consultas do SSB sobre as bases de dados DW1 e DW10. Cada consulta foi executada cinco vezes, e o *cache* limpo após a execução de cada consulta. Todas as consultas foram executadas localmente para inibir a latência da rede. As configurações utilizadas neste experimento foram: (i) *PostgreSQL + JE* utilizou a técnica de junção estrela no PostgreSQL; (ii) *PostgreSQL + VFV* utilizou uma VFV para cada consulta armazenada no PostgreSQL; (iii) *LucidDB + JE* utilizou a técnica junção estrela no LucidDB; (iv) *LucidDB + VFV* utilizou índices *bitmap* de junção previamente construídos sobre cada VFV armazenadas no LucidDB; (v) *MongoDB + DI* utilizou documentos incorporados na coleção *Lineorder*; (vi) *MongoDB + VFV* utilizou uma VFV para cada consulta, eliminando os documentos incorporados; e, (vii) *FastBit + VFV* utilizou índices *bitmap* de junção previamente construídos sobre cada VFV.

Na Fig. 2 são mostrados os tempos médios de execução das consultas para cada configuração em ambas as bases de dados DW1 e DW10. Claramente a configuração *MongoDB + DI* foi a que demonstrou maiores tempos médios nas bases de dados DW1 e DW10. Como melhoramento no processamento de consultas, a configuração *MongoDB + VFV* diminuiu o tempo de resposta das consultas em relação ao *MongoDB + DI*, inferindo que os fatores determinantes foram a minimização da quantidade de atributos e a não adoção de documentos incorporados. Apesar do *MongoDB + VFV* ter melhorado o tempo de execução, esta configuração apresentou tempos proibitivos assim como o uso da técnica da junção estrela, nas configurações *PostgreSQL + JE* e *LucidDB + JE*. O uso de VFVs nas configurações *PostgreSQL + VFV* e *LucidDB + VFV* melhoraram o desempenho em relação a junção estrela. De maneira significativa, a configuração *FastBit + VFV* apresentou os melhores resultados no desempenho do processamento de consultas. Percebe-se então que o uso de índices *bitmap* de junção beneficiou o tempo de resposta no processamento de consultas. Apesar da configuração *LucidDB + VFV* utilizar também índices *bitmap* de junção, os tempos de resposta não foram tão bons quanto o da configuração *FastBit + VFV*. Isso ocorreu devido diferenças de implementações dos referidos índices. Além disso, a cardinalidade de atributos pode ter feito a diferença, uma vez que ao menos um atributo com alta cardinalidade existia em cada consulta (o atributo *lo_revenue* com cardinalidade de 3.345.301 e 5.842.083 nas bases de dados DW1 e DW10, respectivamente).

Em ambas as bases de dados DW1 e DW10 a configuração *FastBit + VFV* apresentou reduções de tempos significativa em relação as outras configurações. Foi calculada a redução de tempo de cada configuração, que determinou o quanto uma configuração se sobrepôs eficientemente sobre outra. Na base de dados DW1 (Fig. 2a), a configuração *FastBit + VFV* foi 31,7% (em relação a Q4.1 da configuração *PostgreSQL + VFV*) a 99,4% (em relação a Q3.4 da configuração *MongoDB + DI*) mais eficiente. Enquanto na base de dados DW10 (Fig. 2b), a configuração *FastBit + VFV* foi 42,4% (em relação a Q4.2 da configuração *PostgreSQL + VFV*) a 99,8% (em relação a Q3.4 da configuração *MongoDB + DI*). Somente na consulta Q4.2, a configuração *LucidDB + VFV* apresentou uma redução de 7,8% em relação a configuração *FastBit + VFV*. Portanto, os índices *bitmap* de junção utilizados no *software* FastBit apresentaram os melhores resultados para processar a maioria das consultas sobre DW quando utilizado visões fragmentadas verticalmente.



(a) Tempo decorrido médio no processamento das consultas sobre a base de dados DW1

(b) Tempo decorrido médio no processamento das consultas sobre a base de dados DW10

Fig. 2. Tempo decorrido médio em segundos por cada configuração para processar as consultas do SSB utilizando as técnicas junção estrela e visões fragmentadas verticalmente sobre a base de dados DW1 e DW10

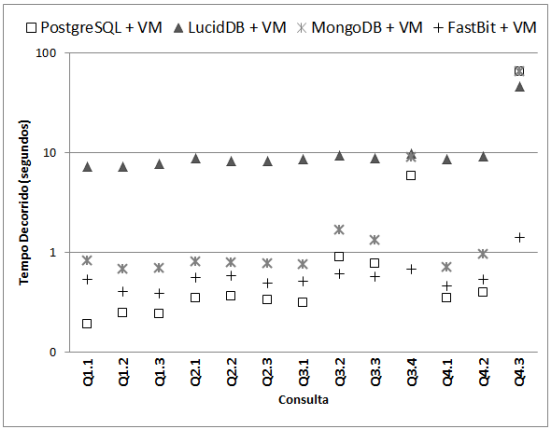
Em relação ao armazenamento, o esquema estrela da base de dados DW1 ocupou uma soma de 1.015 MB no PostgreSQL, 580 MB no LucidDB e 6.813 MB no MongoDB. Enquanto o esquema estrela da base de dados DW10 ocupou uma soma de 9.958 MB no PostgreSQL, 5.847 MB no LucidDB e 70.113 MB no MongoDB. As VFVs na base de dados DW1 ocuparam uma soma de 5.819 MB no PostgreSQL e 10.203 MB no MongoDB, enquanto os índices *bitmap* de junção construídos sobre as visões ocuparam uma soma de 7.640 MB no FastBit e 1.437 MB no LucidDB. No DW10, as VFVs ocuparam uma soma de 63.203 MB no PostgreSQL e 135.162 MB no MongoDB, enquanto os índices *bitmap* de junção construídos sobre as visões ocuparam uma soma de 72.658 MB no FastBit e 14.486 MB no LucidDB. Como resultado, observou-se um volumoso armazenamento do MongoDB em todos os casos, além disso um desempenho proibitivo no processamento de consultas. Em contra partida, o LucidDB mostrou um melhor armazenamento dos índices que o FastBit. Isso se deve a utilização de algoritmos de compressão, porém em sua documentação tais algoritmos não são informados [LucidDB]. Por outro lado, o FastBit proporcionou ganhos nos tempos de respostas em relação ao LucidDB, mesmo requerendo mais espaço de armazenamento.

A aplicação de VFVs na base de dados DW10 ofereceu reduções de tempos em relação a junção estrela. Porém, devido ao alto volume de dados, tais tempos ainda foram proibitivos. Dessa forma, foi investigado o uso de VMs para melhorar o tempo de resposta no processamento de consultas. Utilizaram-se então as seguintes configurações: (i) *PostgreSQL + VM* utilizou visões materializadas armazenadas no PostgreSQL; (ii) *LucidDB + VM* utilizou índices *bitmap* de junção construídos sobre visões materializadas armazenadas no LucidDB; (iii) *MongoDB + VM*: usou visões materializadas armazenadas no MongoDB; e, (iv) *FastBit + VM*: usou índices *bitmap* de junção construídos sobre visões materializadas. Neste teste, todas as consultas do SSB foram executadas localmente e cinco vezes, com o *cache* limpo após a execução de cada consulta.

Na Fig. 3a é mostrado os resultados dos tempos médios. As visões materializadas diminuíram o volume de dados para a execução das consultas, como mostrado na Fig. 3b o número de tuplas de cada VM. O uso de VMs melhorou o desempenho das consultas, porém obteve-se diferença nas visões mais volumosas (Q3.4 e Q4.3). A configuração *FastBit + VM* apresentou reduções de 88,4% a 93% na consulta Q3.4 (configurações *PostgreSQL + VM* e *LucidDB + VM*, respectivamente) e de 96,9% a 97,8% na consulta Q4.3 (configurações *LucidDB + VM* e *MongoDB + VM*, respectivamente). Apesar de o LucidDB utilizar também índices *bitmap* de junção em um volume menor de dados (se comparado às VFVs), o seu resultado não foi proporcional ao do FastBit. Isso se deve a alta compressão utilizada pelo LucidDB que atrasa o tempo de execução das consultas, uma vez que é necessário a descompressão dos índices. Como resultado, os tempos médios da execução das consultas considerando a configuração *FastBit + VM* não teve grande variação. Já a varredura sequencial das VMs no SGBD PostgreSQL foi o que apresentou melhores tempos de respostas médios na maioria das consultas sobre as visões que continham um volume menor de dados. Em relação ao armazenamento, a soma total das VMs ocuparam 4.237 MB no PostgreSQL e 6.783 MB no MongoDB, e os índices *bitmap* de junção construídos sobre as VMs ocuparam uma soma de 5.673 MB no FastBit e 1.364 MB no LucidDB.

3.2 Uso de Memória e de CPU

Este teste experimental objetivou avaliar o consumo de memória e CPU utilizada por cada ferramenta. Foi considerado no teste a consulta Q4.3, sendo a mais custosa constatada na Fig. 3a, executada duas vezes (uma para coletar o consumo de memória e outra para coletar a porcentagem de processamento do CPU) sobre a base de dados DW10. Nesse sentido, as configurações utilizadas na execução das consultas foram as mesmas do teste de avaliação do uso de visões materializadas da seção 3.1: (i) *PostgreSQL + VM*; (ii) *LucidDB + VM*; (iii) *MongoDB + VM*; e, (iv) *FastBit + VM*. O consumo de memória foi medido utilizando a biblioteca *ps_mem.py*. Enquanto o uso porcentual do CPU foi medido utilizando o comando do sistema operacional *ps aux*.



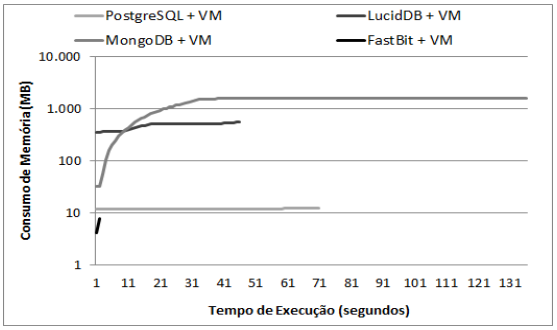
(a) Tempo decorrido médio no processamento das consultas sobre a base de dados DW10

Visão Materializada	Número de Tuplas
Q1.1	7
Q1.2	80
Q1.3	349
Q2.1	35.000
Q2.2	35.000
Q2.3	35.000
Q3.1	4.375
Q3.2	437.500
Q3.3	437.500
Q3.4	4.971.508
Q4.1	4.375
Q4.2	21.875
Q4.3	32.194.206

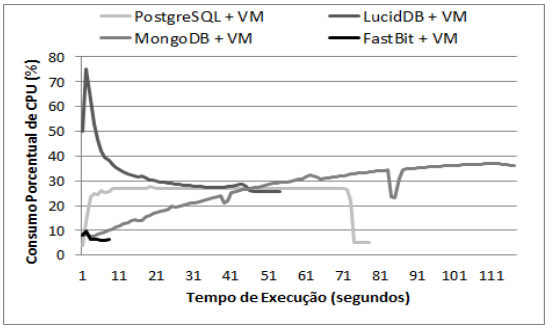
(b) Número de registros de cada visão materializada da base de dados DW10

Fig. 3. Tempo decorrido médio em segundos por cada configuração para processar as consultas do SSB sobre a base de dados DW10, ao utilizar visões materializadas. Número de registros de cada visão materializada composta.

Na Fig. 4a são mostrados os resultados do consumo de memória de cada configuração considerada. A execução da consulta da configuração *MongoDB + VM* apresentou resultados proibitivos, pois consumiu mais memória por um tempo maior (picos de até 1,6 GB) e também demandou mais tempo para ser processada. Já a configuração *LucidDB + VM*, conseguiu reduzir o consumo de memória se comparado a configuração *MongoDB + VM*, porém ainda é altíssimo em relação à *PostgreSQL + VM*. Como resultado principal, a configuração *FastBit + VM* apresentou um excelente consumo de memória aliada a um bom tempo de resposta para processar a consulta. Na Fig. 4b são mostrados os resultados do consumo porcentual de CPU. Novamente, a configuração *MongoDB + VM* exigiu mais tempo para execução da consulta, demandando um custo de processamento de CPU superior as outras configurações durante a maior parte do seu tempo. A configuração *LucidDB + VM* exigiu um grande processamento do CPU no início, devido o início de sua conexão, porém, posteriormente ocorreu uma redução considerável de consumo. Enquanto a configuração *PostgreSQL + VM* apresentou resultados proporcionais, com uma média de consumo de CPU de 27%. Assim, como aconteceu no teste de memória, a configuração *FastBit + VM* exigiu uma carga menor de CPU por um tempo muito menor que as outras configurações.



(a) Consumo de memória



(b) Consumo porcentual de CPU

Fig. 4. Consumo de memória e porcentual de CPU de cada configuração para processar a consulta Q4.3 sobre a base de dados DW10 usando visões materializadas

4. CONCLUSÕES E TRABALHOS FUTUROS

Este artigo apresentou uma investigação experimental utilizando técnicas conhecidas para processar consultas em DW, considerando vários modelos NoSQL e relacional, por meio das bases de dados DW1 e DW10 geradas sinteticamente pelo *Star Schema Benchmark*. Foram realizados experimentos para comparar o tempo de resposta, consumo de memória e consumo porcentual de CPU. Além disso, foi verificado o custo de armazenamento de cada ferramenta avaliada.

Como resultado obtido no tempo médio das execuções de consultas, o *software* FastBit do modelo orientado a coluna por meio de índices *bitmap* de junção garantiu reduções de 25,4% a 99,8% se comparado as outras técnicas utilizadas neste artigo. Adicionalmente, o *software* FastBit apresentou um baixo consumo de memória e consumo porcentual de CPU em relação aos outros modelos. Como pior resultado, o *software* MongoDB apresentou tempos médios proibitivos e altíssimo consumo de memória em relação as outras técnicas no processamento de consultas OLAP sobre DW. Em relação ao armazenamento, o *software* LucidDB do modelo orientado a coluna, apresentou os melhores resultados, e o *software* MongoDB os piores. Porém, devido a alta compressão dos índices *bitmap* de junção, o processamento de consultas foi prejudicado. Dessa maneira, o modelo orientado a coluna do NoSQL obteve melhores resultados no processamento de consultas OLAP bem como no armazenamento dos dados, quando comparado as outras técnicas consideradas neste artigo.

Uma avaliação estendida a ser realizada futuramente acrescentará o modelo orientado a grafos do NoSQL [Han 2011] na avaliação experimental realizada neste artigo. Além disso, a inclusão de dados espaciais nos testes de desempenho também será feita, com a representação de geometrias em DW geográficos para a execução de consultas SOLAP (*Spatial* OLAP) [Siqueira et al. 2011].

Agradecimentos. Os autores agradecem o apoio financeiro das seguintes agências de fomento à pesquisa do Brasil: FAPESP, CNPq e CAPES.

REFERENCES

BAIKOUSI, E. AND VASSILIADIS, P. View usability and safety for the answering of top-k queries via materialized views. In *DOLAP*, pp. 97-104, ACM, New York, 2009.

BONNET, L., LAURENT, A., SALA, M., LAURENT, B. AND SICARD, N. Reduce, You Say: What NoSQL Can Do for Data Aggregation and BI in Large Repositories. In *International Workshop on Database and Expert Systems Applications*, pp.483-488, 2011.

CARNIEL, A. C. AND SIQUEIRA, T. L. L. An OLAP Tool based on the Bitmap Join Index. In *CLEI*, pp. 911-926, 2011.

CATTEL, R. Scalable SQL and NoSQL data stores. Special Interest Group on Management of Data (SIGMOD Record), vol. 39, p. 12-27, ACM, 2010.

GOLFARELLI, M., MAIO, D. AND RIZZI, S. Applying vertical fragmentation techniques in logical design of multidimensional databases. In *DaWaK*, pp. 11-23, Springer, 2000.

HAN, J., HAIHONG, E., LE, G. AND DU, J. Survey on NoSQL database. In *International Conference on Pervasive Computing and Applications (ICPCA)*, pp.363-366, 2011.

KIMBALL, R. AND ROSS, M. *The data warehouse toolkit: the complete guide to dimensional modeling*. John Wiley & Sons, 2002.

LIU, Z., HE, B., HSIAO, H. AND CHEN, Y. Efficient and scalable data evolution with column oriented databases. In *EDBT/ICT'11*, ACM, p. 105-116, 2011.

LUCIDDB. *Data Storage and Access*, http://www.luciddb.org/wiki/LucidDbDataStorageAndAccess.

O'NEIL, P. AND GRAEFE, G. Multi-table joins through bitmapped join indices. SIGMOD Record, vol. 24, pp. 8-11, 1995.

O'NEIL, P., O'NEIL, E., CHEN, X. AND REVILAK, S. The star schema benchmark and augmented fact table indexing. In *TPCTC*, pp. 237–252, 2009.

RÜBEL, O., SHOSHANI, A., SIM, A., STOCKINGER, K., WEBER, G. AND ZHANG, W. M. FastBit: interactively searching massive data. Journal of Physics: Conference Series, vol. 180, 12053, 2009.

SIQUEIRA, T. L. L., CIFERRI, C. D. A., TIMES, V. C. AND CIFERRI, R. R. The SB-index and the HSB-Index: efficient indices for spatial data warehouses. Geoinformatica, vol. 16, no. 1, pp. 165-205, 2011.

STOCKINGER, K. AND WU, K. Bitmap indices for data warehouses. In *Data Warehouses and OLAP*, IRM Press, pp. 157-178, 2006.

XU, L., ZENG, L., SHI, Z., HE Q. AND WANG, M. Research on business intelligence in enterprise computing environment. In *IEEE SMC*, pp.3270-3275, 2007.