UNIVERSIDADE FEDERAL DO ESTADO DO RIO DE JANEIRO
CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA

FRAGMENTAÇÃO DE DATA WAREHOUSES PARA CARGA DE DADOS OPERACIONAIS EM TEMPO REAL

Diego Nolasco Souza Pereira

Orientadores
Asterio Kiyoshi Tanaka
Leonardo Guerreiro Azevedo

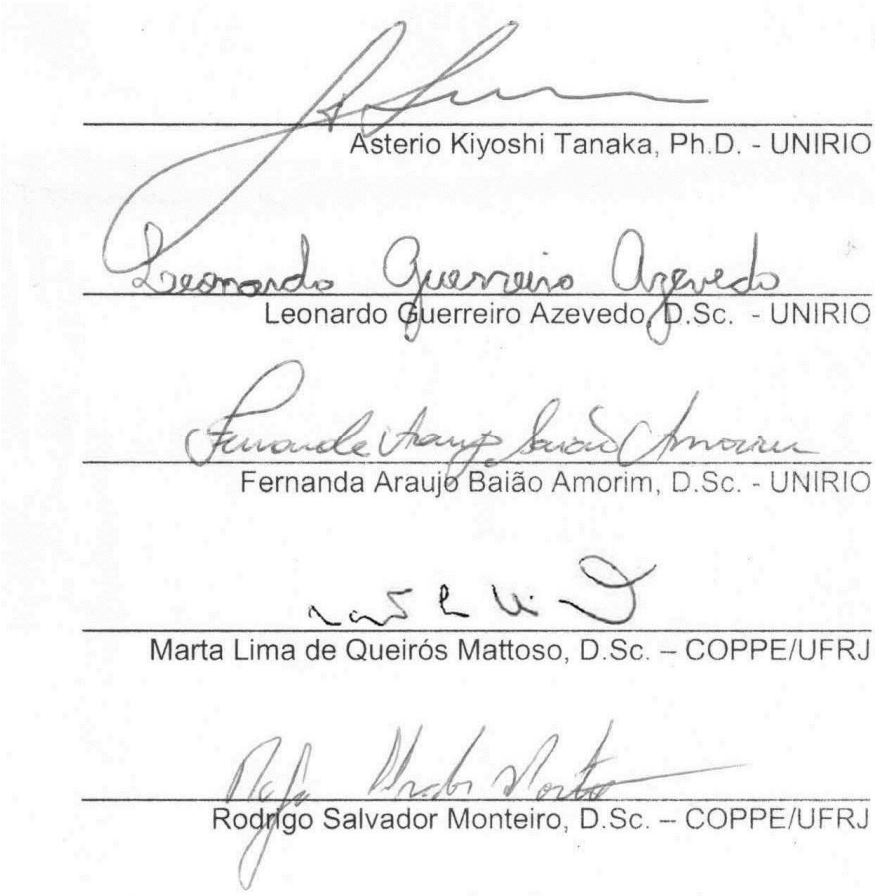
RIO DE JANEIRO, RJ - BRAZIL
SETEMBRO DE 2011

FRAGMENTAÇÃO DE DATA WAREHOUSES PARA CARGA DE DADOS OPERACIONAIS EM TEMPO REAL

Diego Nolasco Souza Pereira

DISSERTAÇÃO APRESENTADA COMO REQUISITO PARCIAL PARA OBTENÇÃO DO TÍTULO DE MESTRE PELO PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA DA UNIVERSIDADE FEDERAL DO ESTADO DO RIO DE JANEIRO (UNIRIO). APROVADA PELA COMISSÃO EXAMINADORA ABAIXO ASSINADA.

Aprovada por:



Asterio Kiyoshi Tanaka, Ph.D. - UNIRIO

Leonardo Guerreiro Azevedo, D.Sc. - UNIRIO

Fernanda Araujo Baião Amorim, D.Sc. - UNIRIO

Marta Lima de Queirós Mattoso, D.Sc. – COPPE/UFRJ

Rodrigo Salvador Monteiro, D.Sc. – COPPE/UFRJ

RIO DE JANEIRO, RJ - BRAZIL
SETEMBRO DE 2011

Pereira, Diego Nolasco Souza.
P436 Fragmentação de data warehouse para carga de dados operacionais em tempo real / Diego Nolasco Souza Pereira, 2011.
viii, 85f.

Orientador: Asterio Kiyoshi Tanaka.
Co-orientador: Leonardo Guerreiro Azevedo.
Dissertação (Mestrado em Informática) – Universidade Federal do Estado do Rio de Janeiro, Rio de Janeiro, 2011.

1. Banco de dados em tempo real. 2. Inteligência empresarial em tempo real. 3. Inteligência do negócio 2.0. 4. Cargas contínuas (Data warehouse). I. Tanaka, Asterio Kiyoshi. II. Azevedo, Leonardo Guerreiro. III. Universidade Federal do Estado do Rio de Janeiro (2003-). Centro de Ciências Exatas e Tecnologia. Curso de Mestrado em Informática. IV.Título.

CDD – 005.74

Dedico este trabalho ao grande amigo Alexandre de Oliveira Martins (In Memoriam). Saudades.

AGRADECIMENTOS

Aos Professores Asterio Kiyoshi Tanaka e Leonardo Guerreiro Azevedo pela excelente orientação fornecida durante toda a realização deste trabalho.

À Professora Fernanda Araújo Baião pelo imenso apoio durante esta jornada.

Ao Professor Sean Wolfgand Matsui Siqueira pelas sugestões e críticas durante os seminários de acompanhamento discente.

À UNIRIO, mais especificamente ao Programa de Pós-Graduação em Informática e seus professores, pelos anos de ensino.

Aos Professores convidados a integrar a banca pela disponibilidade e atenção.

À Caixa Econômica Federal e à Volnei Tonin Zanchin pelo incentivo fornecido.

Aos meus familiares, pets e amigos, em especial a Suzana Botelho Garcez, por compreenderem a minha ausência e sempre me incentivarem.

Ao Luiz Cláudio da Silva Fonseca por me apresentar o mundo de Bancos de Dados.

Agradeço também a todas as pessoas que contribuíram direta e indiretamente para a realização deste trabalho.

PEREIRA, Diego Nolasco Souza. **Fragmentação de Data Warehouses para Carga de Dados Operacionais em Tempo Real.** UNIRIO, 2011. 88 páginas. Dissertação de Mestrado. Departamento de Informática Aplicada, UNIRIO.

RESUMO

Extração, transformação e carga de dados operacionais em tempo real são algumas das características mais proeminentes da próxima geração de ferramentas de Inteligência de Negócio (BI 2.0). Este trabalho apresenta uma proposta para carregar dados operacionais utilizando uma arquitetura de Data Warehouse (DW) que permite tempos de inserção mais rápidos que os de DW tradicionais. Técnicas de distribuição em bancos de dados, como fragmentação horizontal derivada em clusters de computadores *shared nothing* são utilizadas para criar fragmentos especializados nos dados mais atuais e otimizados para permitir a execução de inserções contínuas. Através desta abordagem, o DW pode ser atualizado em tempo quase real a partir das fontes de dados operacionais. Como resultado, consultas no DW são executadas sobre dados em tempo real ou muito próximo disso. Além disso, a execução de cargas contínuas não afeta o tempo de respostas das consultas. Este trabalho também estende o Star Schema Benchmark (SSB) e o utiliza para simular as cargas de dados operacionais em tempo real, a fim de validar e demonstrar a eficiência desta proposta.

Palavras-chave: Data Warehouse em Tempo Real; Inteligência do Negócio em Tempo Real; Inteligência do Negócio 2.0; Cargas Contínuas em Data Warehouses

ABSTRACT

Real-time ETL (Extraction, Transformation and Loading) of enterprise data is one of the foremost features of next generation Business Intelligence (BI 2.0). This work presents a proposal for loading operational data in real time using a Data Warehouse (DW) architecture with faster processing time than current approaches. Distributed database techniques, like derived horizontal fragmentation in shared nothing architecture, are used to create fragments that are specialized in most recent data and optimized to achieve continuous insertions. Using this approach, the DW can be updated near-line from operational data sources. As a result, DW queries are executed over real time data or very close to that. Moreover, continuous loadings do not impact queries response time. In addition, we extended the Star Schema Benchmark to address loading operational data in real time. This benchmark, including the new feature, is used to validate and demonstrate the efficiency of our approach.

Keywords: Real Time Data Warehouse; Real Time Business Inteligence; Business Intelligence 2.0; Continuous Loadings on Data Warehouses

SUMÁRIO

1. Introdução

O presente trabalho de dissertação de mestrado busca permitir a realização de cargas contínuas de dados em Data Warehouses, pré-requisito do processo de Inteligência de Negócio em Tempo Real (Real Time BI). Neste capítulo, temos uma introdução apresentando a motivação do trabalho, seus objetivos e contribuições,além da descrição da estrutura do mesmo.

1.1. Motivação

Inteligência do Negócio (Business Intelligence - BI) é um termo criado no começo da década de 90 pelo Gartner Group (WATSON e WIXON, 2007). Ele representa uma coleção de tecnologias, ferramentas e práticas de *Data Warehousing*, *Data Mining*, processamento analítico, geração de relatórios e visualizações. O objetivo de BI é coletar, integrar, limpar e minerar informações empresariais para auxiliar um pequeno número de especialistas a tomar decisões estratégicas com base em dados históricos (DAYAL *et al*., 2009).

A Figura 1 representa a arquitetura do BI, ilustrando como seus componentes interagem para que ocorra o processo final de auxiliar a tomada de decisão. Esta arquitetura pode ser agrupada em três componentes: ferramentas para coleta e transformação de dados operacionais; servidor de Data Warehouse; e, ferramentas de análise e mineração de dados.

Dados de diversas fontes operacionais, como, por exemplo, dados de sistemas de processamento transacional online (*Online Transaction Processing* - OLTP), são extraídos periodicamente através de ferramentas conhecidas como ETL (*Extract, Transform, Load* – Extração, Transformação e Carga). Estas ferramentas realizam a integração e transformação destes dados para serem carregados em bancos de dados conhecidos como Data Warehouses (DW). No DW, os dados são consultados por aplicações analíticas que facilitam a análise dos mesmos pelo analista responsável pela tomada de decisão (DAYAL *et al*., 2009).

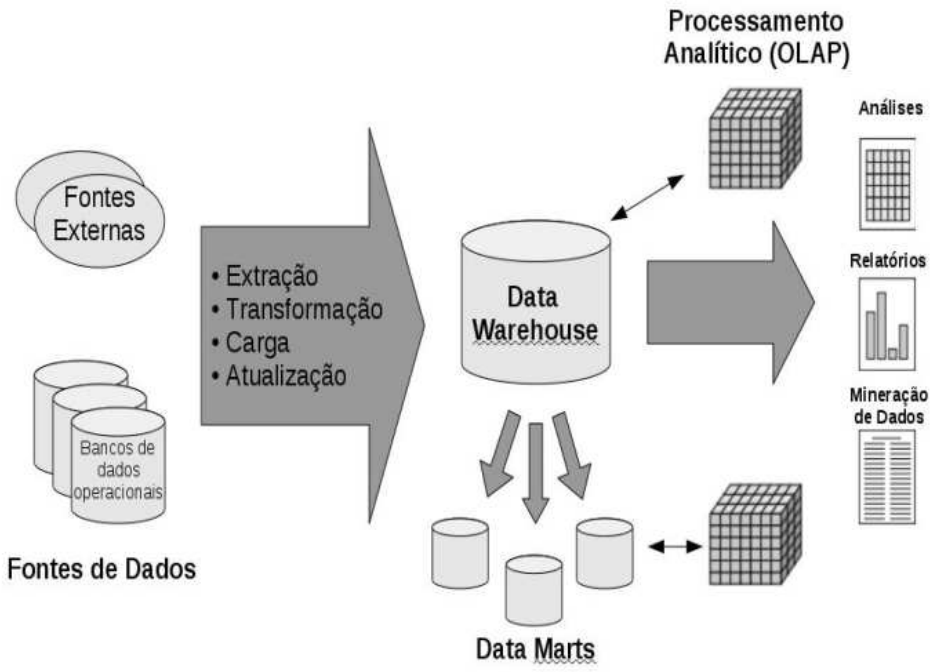


Figura 1 - Componentes do BI - adaptado de (CHAUDHURI *et al.*, 2001)

Data Warehouses são bancos de dados com características bastante peculiares. Por conter dados consolidados e históricos de diversas fontes operacionais, os DW armazenam grandes volumes de dados, geralmente *terabytes* ou mesmo *petabytes* de informações. Além disso, os requisitos das aplicações analíticas que os acessam são bastante distintos das aplicações transacionais, envolvendo consultas *ad hoc* complexas sobre grandes quantidades de registros e realizando diversas operações analíticas, tais como junções e agregações (CHAUDHURI *et al.*, 2001).

Pela popularização e crescente grau de importância das ferramentas de BI, os seus usuários passaram a demandar novas funcionalidades, buscando aprimorar cada vez mais seu processo de tomada de decisão. Além disso, o aparecimento de novos modelos de negócio que possuem características bem distintas dos modelos tradicionais também demanda adaptações na arquitetura de BI tradicional. Estas necessidades desencadearam um processo de evolução da arquitetura para o chamado BI 2.0. As áreas de BI em tempo real, gerenciamento do desempenho de negócio (Business Performance Management) e permeabilidade do BI são algumas destas evoluções pesquisadas atualmente (WATSON e WIXON, 2007).

O gerenciamento do desempenho do negócio busca sumarizar a grande quantidade de dados existentes no Data Warehouse permitindo que o estado do negócio seja visualizado mais rapidamente. Se apoiando em ferramentas como *scorecards* e *dashboards*, tal inovação objetiva exibir de forma rápida e em

pouquíssimas telas o estado atual do negócio e o seu comportamento em relação a metas estabelecidas.

As pesquisas em permeabilidade do BI buscam encontrar formas de disseminar o uso das ferramentas de BI, as quais ainda possuem um elevado grau de complexidade. Tradicionalmente, tais ferramentas possuem um público restrito dentro do local onde são implementadas. Buscar a utilização das informações obtidas com o uso destas ferramentas por um grupo maior dentro da empresa depende de uma maior facilidade no acesso e no uso destas ferramentas, bem como da incorporação das mesmas no cotidiano dos funcionários.

Algumas das linhas adotadas na busca por inovação são o uso de sistemas web, que permitem o acesso às ferramentas, em qualquer lugar com um acesso a internet, de *dashboards* com telas que facilitam o entendimento da informação e a inclusão das ferramentas de BI no processo de negócio de cada uma das áreas da empresa.

O acesso a dados em tempo real refere-se às informações operacionais estarem disponíveis para análise pelas ferramentas de BI o quanto antes, e não apenas no fim do dia, como ocorre no cenário tradicional onde a atualização do DW é realizada fora dos horários produtivos. Tal necessidade é fundamental para a tomada de decisão em negócios como *e-commerce*, bolsas de valores e telecomunicações. Além disso, com a globalização e o uso da *web*, negócios críticos necessitam que as fontes de dados para tomada de decisão estejam disponíveis 24 horas por dia, 7 dias por semana. Assim, as janelas de horário para o processo de carga de um DW se encontram cada vez menores. Dessa forma, um dos requisitos importantes para o sucesso de BI 2.0 é a evolução do processo de ETL e do DW tradicional para suportar um fluxo contínuo de dados, evitando ao máximo qualquer período de indisponibilidade (STODDER, 2007). A complexidade de incluir tal dinamismo em uma arquitetura que opera tradicionalmente em grandes massas de dados históricos, desperta o interesse de vários membros da academia e o desejo de grandes corporações.

1.2. Objetivo

Este trabalho busca solucionar o problema da disponibilização de dados em tempo real nos Data Warehouses (DW), permitindo que as escritas dos dados ocorram em tempos reduzidos e sem prejudicar o desempenho na execução das consultas.

Dentre as áreas relacionadas a este problema, este trabalho foca no armazenamento dos dados no Data Warehouse e no desempenho das consultas

analíticas que os acessam. A solução proposta consiste em uma arquitetura apoiada no uso de técnicas de distribuição e fragmentação nos objetos do DW.

A literatura apresenta alguns trabalhos que tratam esta mesma problemática, porém existem desvantagens associadas aos mesmos que acabam por impedir o seu uso de forma abrangente. Tais desvantagens envolvem principalmente dificuldades no acesso simultâneo de dados históricos e recentes, além de alta complexidade de implementação.

Outras questões relacionadas com este trabalho consistem da obtenção dos dados em fontes operacionais e manipulação dos mesmos. Estes casos são tratados através de abordagens existentes na literatura.

1.3. Contribuições

A principal contribuição deste trabalho é a disponibilização de uma arquitetura de DW capaz de realizar cargas de dados em tempo real, sem impactos nas análises efetuadas em cima de sua massa de dados. Nesta arquitetura, o acesso aos dados históricos e recentes é feito sem que as ferramentas que os manipulam necessitem conhecer sua localidade.

Este trabalho também estende o Star Schema Benchmark (O'NEIL *et al*., 2009) para que as principais características do cenário de DW em tempo real possam ser mensuradas.

1.4. Estrutura

Este dissertação está segmentada em seis capítulos, sendo o primeiro a presente introdução.

No capítulo 2, são apresentados os principais conceitos nos quais o desenvolvimento deste trabalho se apoiou. Temas como distribuição e fragmentação em bancos de dados são descritos no mesmo.

No capítulo 3, o problema alvo desta dissertação, a inteligência de negócio em tempo real, é abordado. Os desafios durante a extração, transformação e armazenamento dos dados em tempo real são introduzidos e as principais propostas existentes para cada uma dessas áreas são apresentadas e avaliadas.

No capítulo 4, a arquitetura proposta neste trabalho é detalhada. As técnicas de fragmentação e distribuição utilizadas são descritas e comparadas com as técnicas tradicionalmente empregadas em Data Warehouses.

No capítulo 5, os experimentos realizados para comprovar a validade da proposta são apresentados. O benchmark SSB e a evolução proposta para o mesmo são utilizados na comparação entre cenários de Data Warehouse, buscando avaliar o comportamento dos mesmos durante a inserção de dados em tempo real. Por fim, são apresentados os resultados obtidos durante a execução dos experimentos, incluindo análises sobre os mesmos.

No capítulo 6, as conclusões obtidas no trabalho são expostas, apontando benefícios e deficiências encontrados e indicando trabalhos futuros que possam aprimorar a proposta descrita por esta dissertação.

2. Fundamentação Teórica

2.1. Arquiteturas de Processamento Distribuído

Sistemas de computação distribuída podem ser definidos como um número de elementos de processamento autônomos, não necessariamente homogêneos, que são interconectados por uma rede de computadores e cooperam entre si para realizarem tarefas designadas (ÖZSU e VALDURIEZ, 1999). Tais sistemas permitem alcançar diversos benefícios, como aumento de disponibilidade, desempenho e escalabilidade (HAYES *et al.*, 1992).

Serviços, como bancos de dados, podem se apoiar neste conceito para obter os benefícios que o processamento distribuído fornece. Assim, sistemas de bancos de dados distribuídos foram desenvolvidos e podem ser definidos como uma coleção de múltiplos bancos de dados logicamente inter-relacionados distribuídos sobre uma rede de computadores (ÖZSU e VALDURIEZ, 1999).

Estes bancos de dados distribuídos podem ser classificados de acordo com os tipos de sistemas multiprocessáveis nos quais eles executam. Sistemas multiprocessáveis são separados de acordo com os dispositivos de memória que os nós de processamento compartilham entre si (ÖZSU e VALDURIEZ, 1999). Estes tipos são classificados por ÖZSU e VALDURIEZ (1999) como *shared nothing*, *shared disk*, *shared memory* e *shared everything*.

- *Shared Nothing*

  Nesta arquitetura cada nó de processamento possui sua própria infra-estrutura de memória e disco. Dessa forma, estes nós necessitam utilizar mecanismos de interconexão de alta velocidade para se comunicarem. A Figura 2 apresenta um modelo da arquitetura.

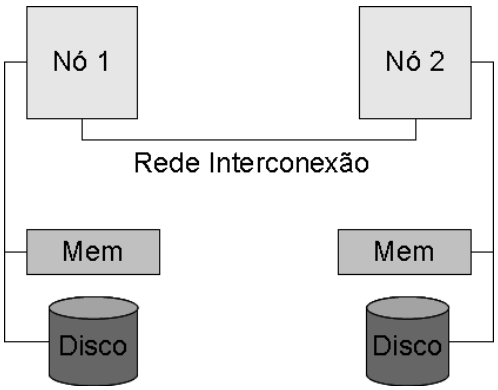


Figura 2 - Arquitetura Shared Nothing

- *Shared Disk*

  Também definido como fracamente acoplado (*loosely coupled*), neste cenário, cada nó de processamento possui sua própria infra-estrutura de memória, porém o disco de ambos é compartilhado. Assim, cada nó consegue acessar todos os dados existentes, permitindo que operações SQL executadas em qualquer um deles acessem todos os dados. A Figura 3 apresenta um modelo da arquitetura.

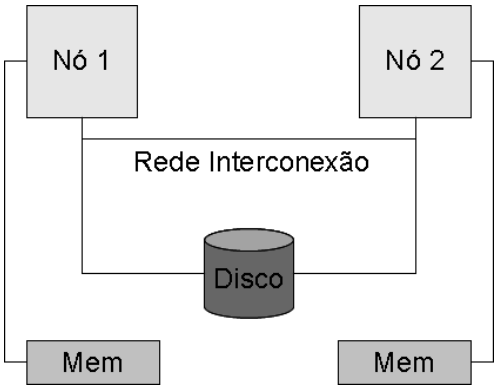


Figura 3 - Arquitetura Shared Disk

- *Shared Memory*

  Também definido como fortemente acoplado (*tightly* coupled), neste cenário, cada nó de processamento possui sua própria infra-estrutura de discos, porém a memória de ambos é compartilhada. Com isso, os nós podem se comunicar sem trocar mensagens via rede. A Figura 4 apresenta um modelo da arquitetura.

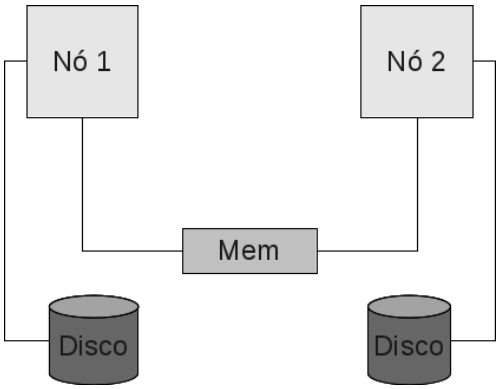


Figura 4 – Arquitetura *Shared Memory*

- *Shared Everything*

  Neste cenário, os nós acessam o disco e a memória de forma compartilhada. Assim, cada nó possui controle completo sobre o banco de dados como um todo, sejam os dados ou *buffers* de memória. A Figura 5 apresenta um modelo da arquitetura.

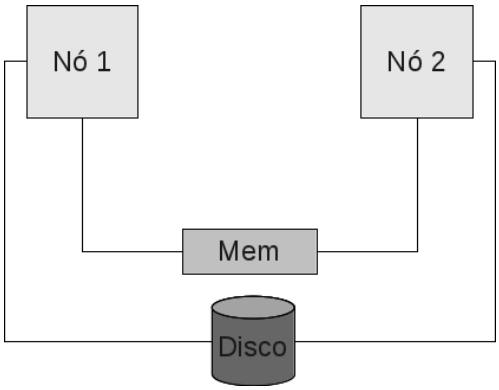


Figura 5 - Arquitetura *Shared Everything*

Produtos disponíveis no mercado geralmente se baseiam nas arquiteturas *shared nothing* (por exemplo, Teradata[1], MySQL Cluster[2], pgpool[3]) e *shared disk* (por exemplo, Oracle RAC[4], DB2 com Parallel Sysplex[5]). Comparando estas duas

[1] http://www.teradata.com/WorkArea/DownloadAsset.aspx?id=4650
[2] http://www.mysql.com/products/cluster/
[3] http://pgpool.projects.postgresql.org/pgpool-II/doc/pgpool-en.html
[4] http://www.oracle.com/us/products/database/options/real-application-clusters/index.html
[5] http://www-03.ibm.com/systems/z/advantages/pso/

arquiteturas, as *shared nothing* necessitam de um esforço inicial e contínuo maior pela necessidade de avaliar a melhor forma de fragmentação, enquanto que na *shared disk* não há esforço nenhum nesse sentido. Porém, quando a fragmentação é bem realizada, o desempenho obtido na *shared nothing* tende a ser superior devido ao acesso exclusivo aos discos por cada nó (HOGAN, 2009).

Segundo ÖZSU e VALDURIEZ (1999) a arquitetura *shared nothing* é a que mais se assemelha a sua proposta de ambiente de distribuição de bancos de dados. Um dos seus argumentos é a de que esta arquitetura possibilita suportar heterogeneidade de sistemas operacionais e *hardware*, se assemelhando aos cenários propostos na computação distribuída. Seguindo esta linha, as discussões sobre distribuição em bancos de dados deste trabalho abordarão técnicas aplicadas a esta arquitetura.

2.2. Distribuição em Bancos de Dados

Bancos de dados distribuídos são bancos de dados onde a gerência dos dados está distribuída entre diversas máquinas interconectadas através de uma rede de computadores, ou seja, os dados se encontram fragmentados e replicados em diversos servidores interconectados, os quais podem ser parte de um cluster ou de sites geograficamente distantes (RISCH, 2009).

Técnicas para bancos de dados distribuídos surgiram no final da década de 1970 em conjunto com os clusters de bancos de dados *shared nothing* (TAN, 2009b), sendo o Ingres e o System R softwares pioneiros de gerência destes tipos de bancos de dados. Porém, estes software foram desenvolvidos considerando a lentidão das redes de computadores da época, fator que limitava bastante a evolução da tecnologia. Com a evolução das tecnologias de redes na década de 1990 e dos clusters de computadores, a tecnologia de bancos de dados distribuídos pôde se desenvolver bastante (ELNIKETY, 2009).

Os dois métodos fundamentais utilizados na distribuição de bancos de dados em arquiteturas *shared nothing* são a fragmentação e a replicação. A fragmentação consiste em dividir um conjunto de dados em diversos segmentos com dados exclusivamente alocados a eles, enquanto que a replicação consiste em se obter cópias idênticas do conjunto de dados alvo.

A fragmentação permite que o administrador do banco de dados (DBA) especifique em quais nós o gerenciador deve alocar determinado grupo de dados, permitindo um aumento de desempenho na execução de consultas e atualizações,

pois permite o acesso paralelo nos fragmentos distribuídos. Maiores detalhes são apresentados na Seção 2.3.

A replicação permite ao DBA solicitar ao gerenciador que coloque o mesmo conjunto de dados em mais de um nó, acelerando a consulta aos dados enquanto aumenta os custos de atualização.

A alocação de fragmentos ou a escolha de replicação dos mesmos são feitas de acordo com critérios de otimizações estabelecidos, tais como, aumento da taxa de transferência (*throughput*), diminuição do tempo de resposta, diminuição do custo de execução de uma operação (ELNIKETY, 2009).

Os principais benefícios obtidos com a distribuição são (ELNIKETY, 2009):

- Alocação de dados em sites mais próximos de onde os mesmos são mais freqüentemente consultados, diminuindo o tempo gasto com comunicações.
- Paralelismo intra-consultas, onde uma consulta é quebrada em sub-consultas que podem ser executadas simultaneamente.
- Paralelismo inter-consultas, onde diversas consultas podem ser executadas simultaneamente.
- Aumento da disponibilidade dos dados, pois com a falha de um dos nós os dados contidos nos outros podem continuar sendo acessados.

2.2.1. Sistemas Gerenciadores de Bancos de Dados Distribuídos

Um Sistema Gerenciador de Bancos de Dados Distribuídos (SGBDD) gerência bancos de dados em múltiplos sites, cada um com fragmentos dos dados. Estes fragmentos podem ser replicados entre os sites, variando entre um banco de dados completamente replicado, onde cada site possui uma cópia completa dos dados, e um banco de dados completamente fragmentado, onde os dados são divididos em fragmentos e cada fragmento é exclusivamente associado a um site. Assim, os SGBDDs necessitam gerenciar metadados que indiquem qual site é responsável por qual parte dos dados (ELNIKETY, 2009). O principal objetivo destes softwares é o de esconder a distribuição de dados, fazendo com que os bancos se comportem como se fossem um só para o usuário do banco de dados. Assim, consultas e atualizações a dados distribuídos são resolvidos pelo SGBDD em operações de dados nos nós afetados, dando ao usuário a impressão de estar usando apenas um banco de dados.

Os principais desafios dos SGBDDs são: como realizar a distribuição e alocação dos dados; como atualizar todas as cópias dos dados distribuídos quando os mesmos sofrerem modificações; e, como executar consultas que acessam dados alocados em múltiplos sites.

SGBDDs geralmente possuem os seguintes componentes (CERI e PELAGATTI, 1984):

- Gerenciador de banco de dados local
  - SGBD centralizado responsável pelo banco de dados local a ele associado.
- Componente de comunicação de dados
  - Responsável pela comunicação entre os nós, como, por exemplo, envio de respostas de sub-consultas executadas.
- Dicionário de dados
  - Possui os metadados relacionados aos critérios de distribuição de dados existentes e os nós associados.
- Componente de banco de dados distribuídos
  - Atua como o gerenciador do banco de dados distribuídos realizando a interação com os outros componentes;
  - Assegura o correto funcionamento do banco de dados distribuídos, resolvendo situações como controle de *deadlocks* entre os nós e controle de atomicidade de transações que ao serem quebradas podem gerar inconsistências.

2.2.2. Transparência de Bancos de Dados Distribuídos

O termo transparência é utilizado em bancos de dados distribuídos para indicar que algo é resolvido de forma oculta para o usuário, evitando que ele precise tomar conhecimento de como a operação é realizada. Tal conceito é justamente o oposto do utilizado em diversas outras áreas, onde transparência aponta visibilidade completa para o usuário de como determinado processo ocorre. Seguindo a definição do conceito utilizada pelos bancos de dados distribuídos, os mesmos podem ser classificados de acordo com o tipo de transparência fornecida (ÖZSU e VALDURIEZ, 1999) (RISCH, 2009):

- Transparência para esquemas:

  Neste tipo de transparência, o SGBDD decide automaticamente como distribuir os dados entre os nós se comportando como um banco de dados único até mesmo para o DBA, o qual cria os esquemas lógicos sem considerar os critérios de distribuição, como se estivesse em um banco de dados normal. Alguns SGBDDs com este nível de transparência permitem uma flexibilização para que o DBA possa realizar ajustes finos nos critérios de distribuição preparados, compensando fatores como a latência de acesso aos dados quando os nós estão distantes geograficamente.

- Transparência para consultas:

  Neste tipo de transparência, a distribuição dos dados não é refletida nas consultas de usuários, ou seja, as consultas continuam sendo elaboradas como se existisse um único banco de dados. Tais consultas são traduzidas transparentemente em sub-consultas executadas nos nós onde os dados se encontram e os resultados parciais são encaminhados entre os nós até que o resultado completo possa ser encaminhado para o usuário. Este tipo de transparência é fundamental para um SGBDD, pois a sua implementação de forma manual pela aplicação é altamente sujeita a falhas e a re-trabalho em reorganizações do projeto da distribuição dos dados.
- Transparência para atualizações:

  Neste tipo de transparência, atualizações nos dados são feitas sem que os critérios de distribuição precisem ser conhecidos. O SGBDD recebe as mesmas e as propaga para todos os nós necessários.

Os SGBDD tradicionais oferecem ao menos as transparências para consultas e atualizações.

2.3. Fragmentação de Bancos de Dados Relacionais

A fragmentação de bancos de dados relacionais consiste em subdividir um conjunto de dados em pedaços menores de tal forma que o conjunto de dados original possa ser reconstruído a partir da união dos fragmentos.

Existem diversos motivos para o uso desta técnica, sendo o mais usual o fato de que aplicações tipicamente acessam apenas subconjuntos dos dados. Dessa forma, ao realizar operações em segmentos distintos, a quantidade de operações concorrentes pode ser aumentada. Nos casos onde as aplicações acessam grandes volumes de dados, a fragmentação permite aumentar o nível de paralelismo das consultas através da quebra das mesmas em sub-consultas (TAN, 2009a).

Existem basicamente três técnicas de fragmentação: horizontal primária, horizontal derivada e vertical. Tais técnicas podem ser utilizadas isoladamente ou em conjunto, aplicando uma técnica sobre os fragmentos gerados pela execução de outra técnica. Este último cenário é conhecido como fragmentação híbrida (TAN, 2009a).

2.3.1. Fragmentação Horizontal Primária

Na fragmentação horizontal primária, cada fragmento gerado é essencialmente um subconjunto de tuplas da relação original, sendo geralmente definido como uma seleção desta relação. Uma boa fragmentação horizontal primária possui tipicamente

três propriedades: completude, disjunção e reconstrutibilidade. Completude indica que toda tupla da relação original precisa estar associada a um fragmento. Disjunção indica que as tuplas da relação original devem estar associadas exclusivamente a um fragmento. Reconstrutibilidade indica que a relação original pode ser reconstruída simplesmente pela união de todos os fragmentos gerados. A Figura 6 ilustra este tipo de fragmentação, nela a tabela Empregados é fragmentada horizontalmente a partir da coluna Localização.

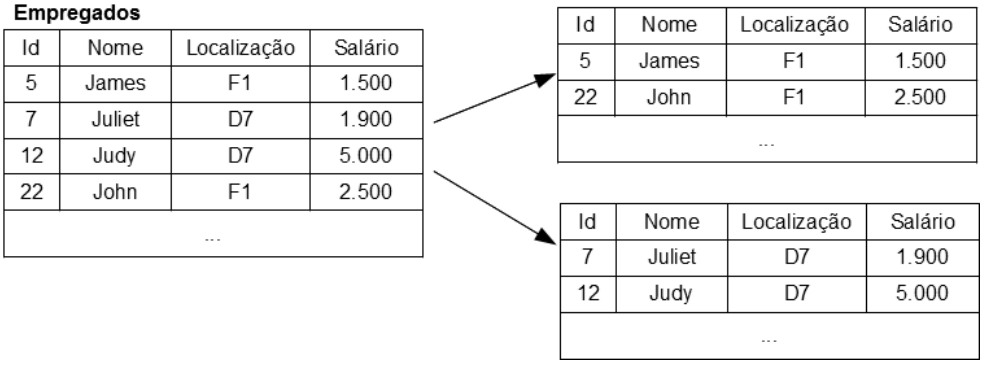


Figura 6 – Exemplo de fragmentação Horizontal Primária

2.3.2. Fragmentação Horizontal Derivada

Na técnica de fragmentação horizontal derivada, uma relação é fragmentada de acordo com critérios definidos em outra relação. A relação fragmentada recebe o nome de membro e a relação utilizada como critério para fragmentação é chamada de proprietária. Cada fragmento é obtido através de uma semi-junção entre a relação membro e o fragmento que atende ao critério escolhido da relação proprietária. Para que esta técnica possa atender à propriedade de completude, o conjunto de atributos utilizados como critério para a fragmentação na relação membro deve compor chaves estrangeiras para os atributos na relação proprietária. Além disso, para que a propriedade de disjunção possa ser alcançada, os atributos utilizados como critério da fragmentação devem compor a chave primária da relação proprietária. A Figura 7 ilustra este tipo de fragmentação, nela a tabela Projetos é fragmentada horizontalmente de forma derivada a partir da coluna Local pertencente à tabela Empregados. A relação entre as tabelas ocorre através da coluna Id, chave primária da tabela Empregados, e da coluna Resp. da tabela Projetos.

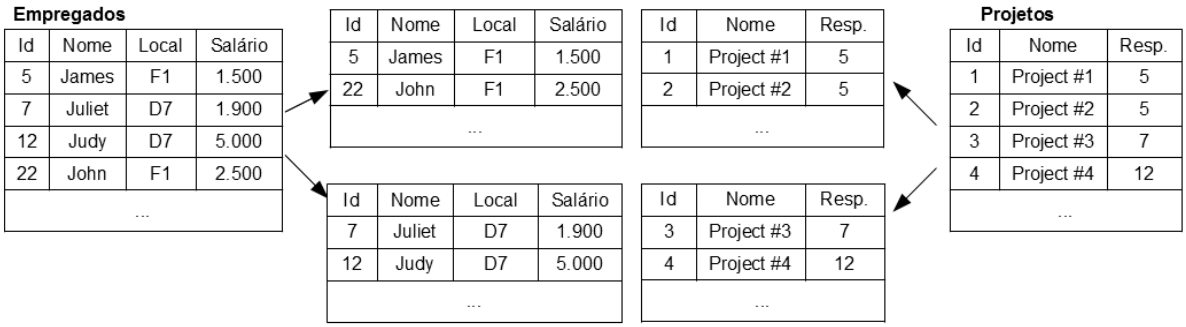


Figura 7 – Exemplo de fragmentação Horizontal Derivada

2.3.3. Vertical

A fragmentação vertical divide uma relação em um conjunto de fragmentos que possuem um subconjunto de atributos da mesma, sendo cada fragmento uma projeção da relação original. Para que esta técnica possua a propriedade de reconstrutibilidade é recomendável que os atributos da chave primária da relação existam em cada um dos fragmentos. A Figura 8 ilustra este tipo de fragmentação, nela a tabela Empregados é fragmentada verticalmente buscando separar os valores da coluna Salário, dos valores das colunas Nome e Localização.

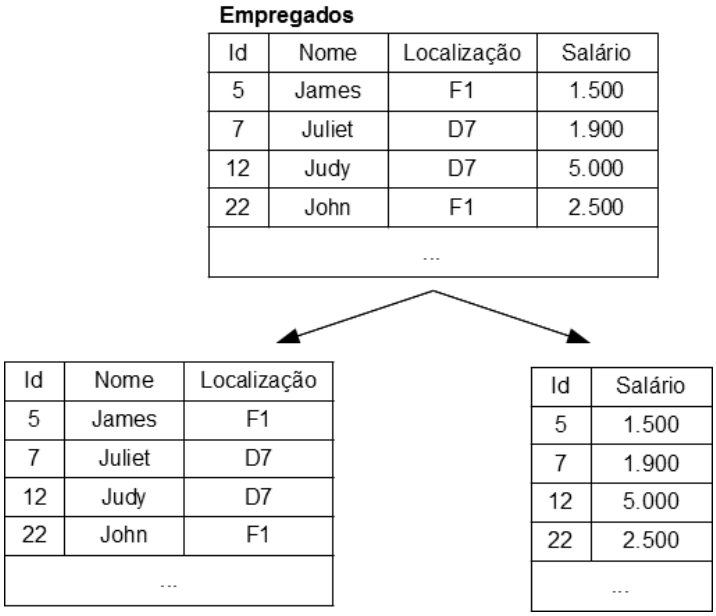


Figura 8 – Exemplo de fragmentação Vertical

2.3.4. Híbrida

A fragmentação híbrida é uma classificação dada quando técnicas de fragmentação são aplicadas em cima de dados já fragmentados por outras técnicas, caracterizando a aplicação em conjunto das técnicas descritas acima. A Figura 9 ilustra este tipo de fragmentação, nela a tabela Empregados é fragmentada horizontalmente na coluna Localização. Posteriormente, o fragmento com valor de Localização D7 é fragmentado verticalmente buscando separar os valores das colunas Salários, dos valores das colunas Nome e Localização.

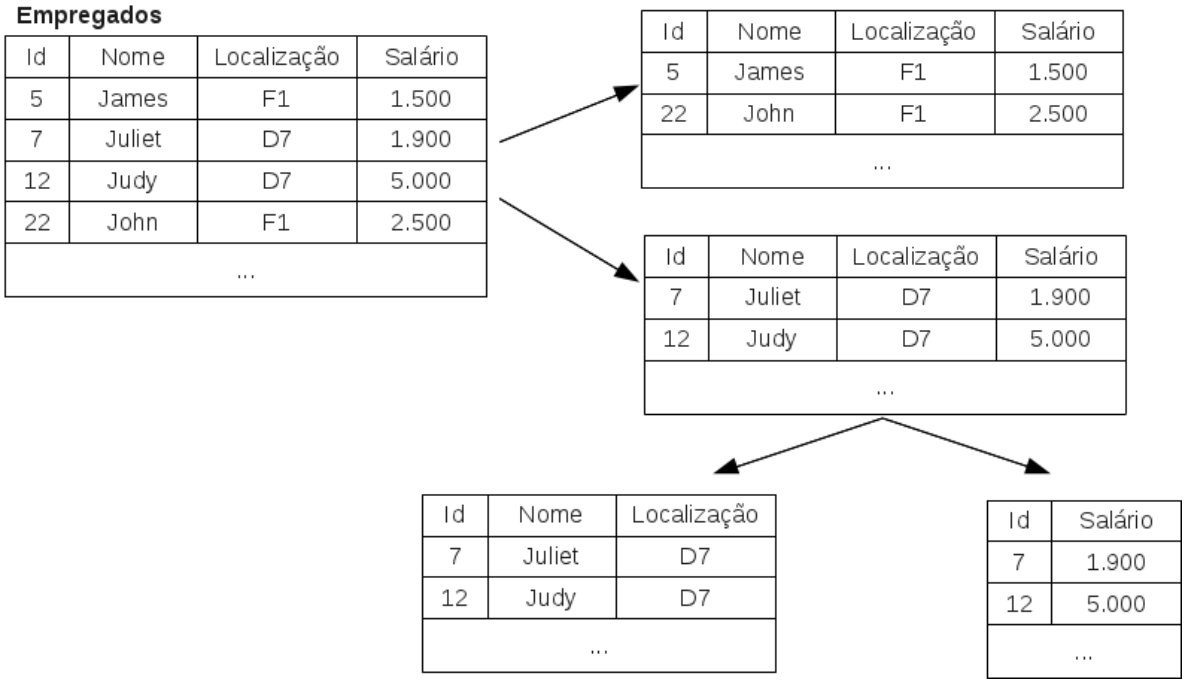


Figura 9 – Exemplo de fragmentação Híbrida

3. Inteligência do Negócio em Tempo Real

Inteligência do Negócio em Tempo Real é uma das evoluções mais desafiadoras que compõem o cenário de Inteligência do Negócio 2.0. Este termo é utilizado para definir soluções de BI que permitem a análise dos dados operacionais com intervalos de latência bastante reduzidos. Tal termo engloba não apenas cenários cuja latência é efetivamente zero, como inicialmente pode se pressupor, mas também cenários cuja latência alcançada fica na escala de minutos (KIMBALL e CASERTA, 2004).

Tradicionalmente, soluções de BI executam processos de ETL em fontes de dados operacionais diariamente, em horários de baixa utilização, a fim de carregar tais dados no DW para análise por suas ferramentas. Assim, os desafios da Inteligência do Negócio em Tempo Real surgem na tentativa de redução da latência deste processo. Estes desafios podem ser agrupados em três áreas (INMON e STRAUSS, 2008, KIMBALL e CASERTA, 2004): Extração dos Dados, Transformação dos Dados e Armazenamento dos Dados.

A **extração dos dados** geralmente ocorre em fontes do tipo OLTP (Online Transactional Processing), as quais são otimizadas para realização de grandes quantidades de pequenas escritas (isto é, considerando poucos dados) em curtíssimo tempo. A leitura destas fontes para obtenção dos dados para o DW, durante os períodos de maior atividade, pode gerar impactos que elevem bastante o tempo de escrita das aplicações transacionais, impactando a execução do negócio da empresa.

As ferramentas de **transformações de dados** são construídas para a arquitetura tradicional de ETL, com execuções pouco freqüentes em grandes massas de dados. No cenário de BI 2.0 as transformações possuem características opostas, sendo executadas constantemente em massas de dados pequenas.

As soluções tradicionais para **armazenamento dos dados** utilizam diversas técnicas, tais quais índices e visões materializadas, para diminuir o tempo das consultas realizadas nos DWs. Estas técnicas afetam o desempenho de inserções, as quais acabam sendo realizadas em lotes e fora do horário de execução das consultas.

Entretanto, no cenário de BI 2.0, a realização de inserções constantes no DW se torna necessária, tornando o seu desempenho um ponto adicional de preocupação.

A seguir são apresentadas soluções encontradas na literatura para cada uma destas três áreas, porém vale ressaltar que o foco da proposta é atuar na área de armazenamento de dados.

3.1. Extração e Transformação em Tempo Real

A construção de uma solução de ETL para atender ao cenário de inteligência de negócio em tempo real demanda uma série de pré-requisitos, tais como: classificar os objetivos do negócio; entender um extenso conjunto de tecnologias; ter conhecimento de abordagens utilizadas com sucesso por outros; além de se ter uma grande flexibilidade e criatividade no desenvolvimento (KIMBALL e CASERTA, 2004).

Algumas tecnologias estão sendo desenvolvidas para atender aos requisitos do ETL em tempo real, porém cada uma possui características distintas, devendo a sua escolha ser avaliada de acordo com a necessidade do negócio onde o cenário de BI em Tempo Real está sendo implantado. Esta seção descreve algumas destas técnicas.

3.1.1. *Microbatch* ETL

O *Microbatch* ETL é bastante similar ao ETL convencional, tendo como diferencial a execução muito mais freqüente, geralmente de hora em hora. Devido a esta execução muito mais freqüente, tais ferramentas necessitam de mecanismos de controle de tarefas, agendamento, dependências e resolução de erros mais complexos do que os existentes nas ferramentas de ETL tradicionais. Além disso, estas ferramentas também dependem de mecanismos de detecção de novas inserções e atualizações de registros nas fontes de dados operacionais como, por exemplo:

- *Timestamps*: Verificando as marcas de tempo utilizadas pelas fontes operacionais para cadastrar novos registros pode-se verificar quais necessitam ser transportados.
- Tabelas de *log*: através do uso de gatilhos nos bancos de dados operacionais pode-se cadastrar em uma tabela de *log* identificadores para os registros atualizados ou inseridos.
- *Log* de Transação do SGBD: *Logs* de transação gerados por bancos de dados como mecanismos de auxílio para *backup* e *recovery* podem ser lidos para que as transações cadastradas nos mesmos sejam recriadas e transportadas.

- *Sniffers* de Rede: Através da monitoração do fluxo de rede, pode-se filtrar o tráfego relacionado a informações desejadas e capturá-las para envio ao Data Warehouse.

A Figura 10 apresenta a arquitetura do *Microbatch* ETL. Nela, mecanismos de detecção de mudanças monitoram as aplicações A e B, acumulando as alterações realizadas para transportá-las posteriormente ao Data Warehouse de Tempo Real.

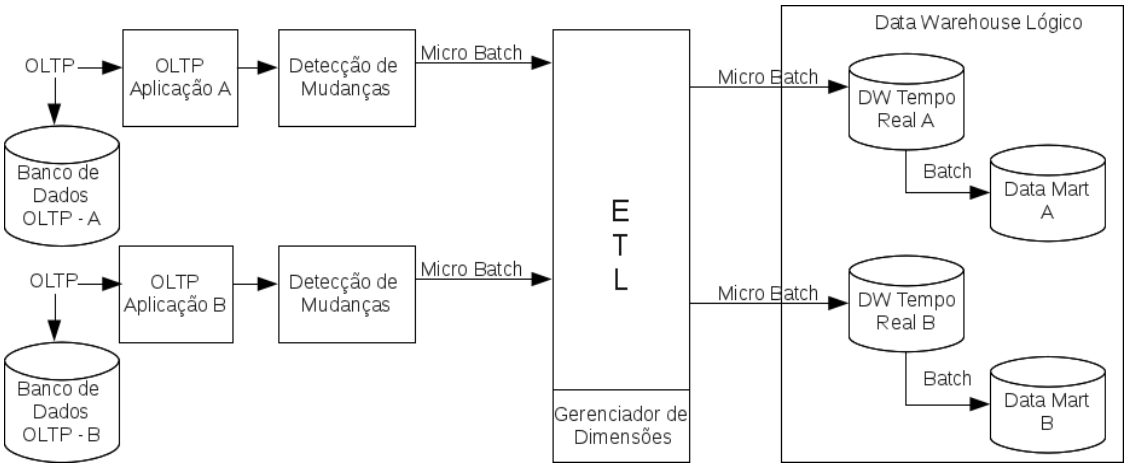


Figura 10 - Arquitetura do Microbatch ETL - adaptado de (KIMBALL e CASERTA, 2004)

3.1.2. *Enterprise Application Integration*

A implementação desta tecnologia é a mais complexa dentre as abordagens de ETL em tempo real. *Enterprise Application Integration* (EAI) tipicamente consiste na construção de um conjunto de adaptadores e intermediadores que utilizam mensagens para comunicar as transações do negócio que estão ocorrendo. Enquanto os adaptadores são responsáveis por criar ou executar as mensagens, os intermediadores devem encaminhar as mensagens recebidas de acordo com um conjunto de regras pré-estabelecidos.

Além do uso de EAI para propagar as transações ocorridas, o mesmo também facilita a realimentação das fontes operacionais com informações corrigidas no Data Warehouse.

A Figura 11 apresenta uma arquitetura de EAI que atende ao requisito de BI em tempo real. Nela, cada componente da arquitetura de BI possui um adaptador que se comunica com o elemento intermediador, responsável por endereçar corretamente as mensagens recebidas.

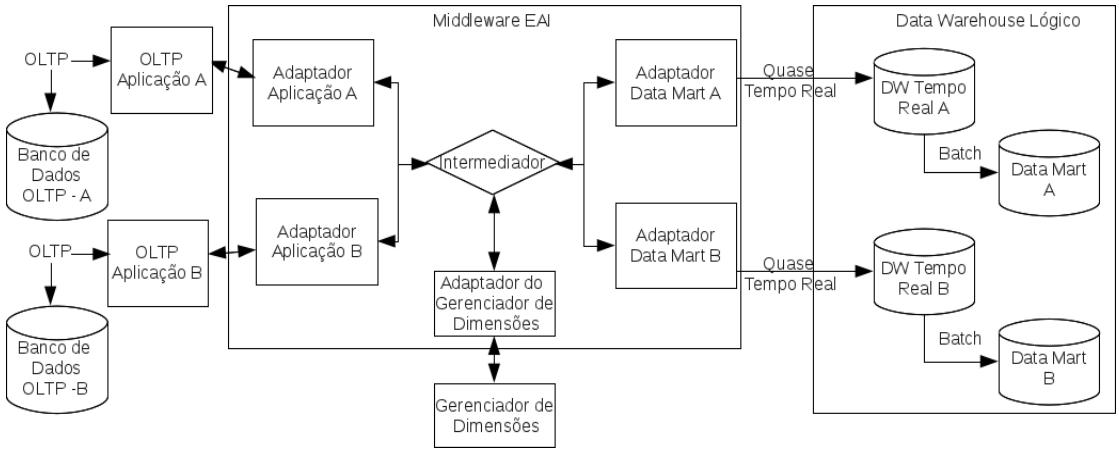


Figura 11 - Arquitetura EAI para BI em tempo real - adaptado de (KIMBALL e CASERTA, 2004)

3.1.3. *Capture, Transform* e *Flow*

Capture,Transform e Flow (CTF) é uma nova categoria de ferramentas de integração de dados desenvolvida para simplificar a movimentação de dados em tempo real entre bancos de dados. As transformações realizadas nos dados pelo CTF são bastante básicas em comparação com as realizadas por ferramentas ETL. Dessa forma, ferramentas CTF tendem a usar uma área intermediária, chamada de Área de *Staging*, onde posteriormente os dados são extraídos por ferramentas ETL que realizam as tarefas de transformação mais complexas nos dados antes de inserir no Data Warehouse.

Na Figura 12, a arquitetura do CTF é apresentada. Nela, os dados são extraídos das aplicações A e B pelas ferramentas CTF e inseridos em respectivas áreas de *staging*. Posteriormente, ferramentas de ETL transformam e transportam os dados destas áreas de *staging* para o Data Warehouse.

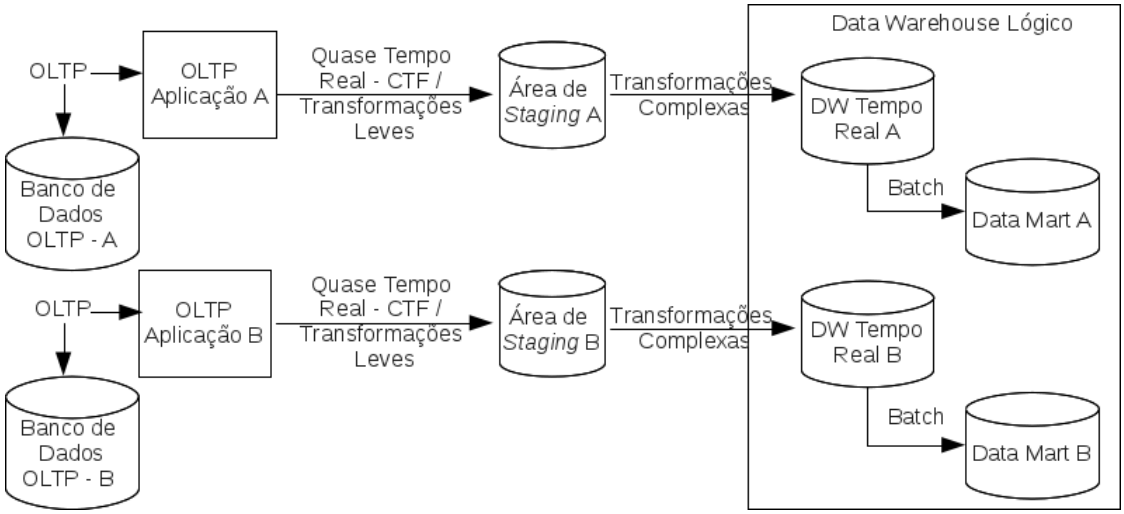


Figura 12 - Arquitetura de CTF - adaptado de (KIMBALL e CASERTA, 2004)

3.1.4. *Enterprise Information Integration*

Enterprise Information Integration (EII) é uma categoria de software que objetiva auxiliar organizações com a disponibilidade rápida de funcionalidades *real-time* em seu sistema de inteligência de negócio. EII se comporta de maneira semelhante ao ETL, porém o destino dos dados não é Data Warehouses, mas sim relatórios ou planilhas. Quando o analista do negócio necessita das informações atualizadas ele aciona o EII que gera uma série de consultas para recarregar o relatório ou planilha alvo. Dessa forma, o EII permite que os relatórios tenham informações com tempo de latência igual a zero. Porém, por não possuir dados históricos, os relatórios apenas podem ser preenchidos com os dados disponíveis nos bancos de dados operacionais. Uma alternativa para suporte a dados históricos seria a de manter paralelamente um Data Warehouse e defini-lo como fonte para o EII, assim como os dados operacionais (KIMBALL e CASERTA, 2004).

A Figura 13 ilustra uma arquitetura de EII. Nela, o servidor de EII atende requisições dos usuários permitindo análises diretas sobre dados de diversas fontes do ambiente do negócio.

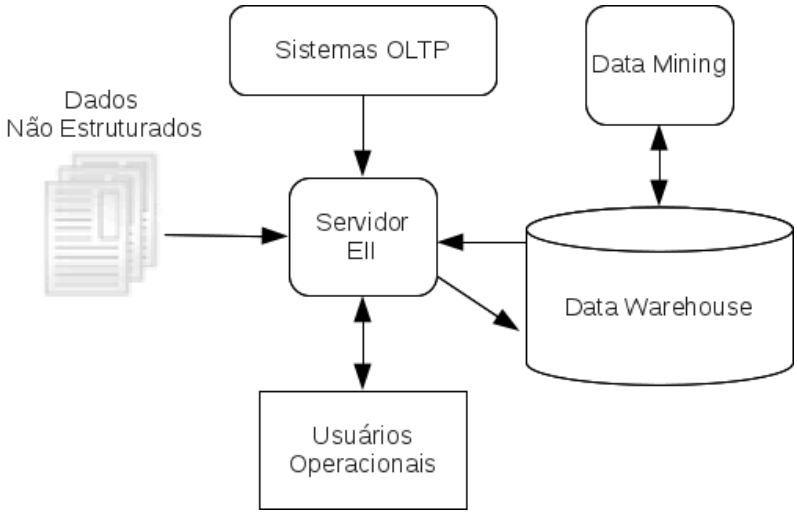


Figura 13 – Arquitetura de EII - adaptado de (HATAMI, 2007)

3.1.5. Comparações Entre as Ferramentas

KIMBALL e CASERTA (2004) classificam os mecanismos para ETL em tempo real através dos seguintes pontos: Suporte a dados históricos; Nível de complexidade permitido às tarefas de integração de dados; Latência necessária para disponibilidade dos dados do negócio.

A Tabela 1 apresenta uma comparação seguindo tais pontos. Ao analisar a mesma percebe-se que geralmente existe uma troca entre a latência para disponibilidade dos dados (coluna Atualidade dos Dados) e o nível de complexidade

permitido para análises futuras (coluna Complexidade Permitida na Integração de Dados).

Apenas verificando tais pontos, a solução EAI desponta como ideal para o desafio de ETL em tempo Real ao suportar baixos níveis de latência sem afetar sua capacidade de integração de dados. Porém, seu ponto fraco reside na grande complexidade e nos elevados custos para sua implementação. Adicionalmente, tal ferramenta pode ser ineficiente ao lidar com a replicação de altos volumes de transação. Tal fato ocorre já que cada transação é empacotada como uma mensagem EAI, o que gera uma sobrecarga nas comunicações.

Assim, empresas que enfrentam desafios de integrações de dados deveriam adotar ferramentas como *Microbatch* ETL e EAI. Já empresas cujo foco seja latências extremamente baixas devem buscar ferramentas como EII, CTF e EAI.

Tabela 1 - Comparativo de Técnicas de ETL em tempo Real - adaptado de (KIMBALL e CASERTA, 2004)

	Suporte a Dados Históricos	Complexidade Permitida na Integração de Dados	Atualidade dos Dados
ETL Tradicional	Sim	Alta	1 Dia
Microbatch ETL	Sim	Alta	< 1 Hora
EAI	Sim	Alta	< 1 Min
CTF	Sim	Moderada	< 15 Min
EII	Não	Moderada	< 1 Min
EII + DW Estático	Sim	Moderada	< 1 Min

3.2. Armazenamento em Tempo Real

Evoluir o Data Warehouse para permitir a realização de cargas de dados contínuas, simultaneamente às consultas complexas tradicionalmente executadas nele, tem sido um grande foco de pesquisa desde o surgimento da necessidade da Inteligência de Negócios em Tempo Real (STODDER, 2007, WATSON e WIXON, 2007).

Nesta seção são apresentadas algumas das propostas encontradas na literatura que buscam resolver este problema.

3.2.1. Ordenação das Operações Enviadas ao DW

Algumas propostas buscam solucionar o problema de armazenamento de dados em tempo real através de mecanismos para ordenar as operações enviadas ao DW, tais quais THIELE *et al.* (2007), SHI *et al.* (2009) e MARTINS (2009).

THIELE *et al.* (2007) propõe um serviço chamado de *Workload Balancing by Election* (WINE). Este serviço se baseia na premissa de que os usuários ao acessarem o DW podem escolher entre tempos de resposta curtos, definido como qualidade de serviço (*Quality of Service* - QoS), ou dados mais recentes, definido como qualidade de dados (*Quality of Data* - QoD). Cada consulta efetuada pelos usuários deve especificar a sua escolha entre tais fatores, sendo considerada como um voto pelo serviço para decidir se o mesmo irá priorizar consultas ou atualizações. Por fim, um componente chamado *Workload Balancing Unit* (WBU) é responsável por gerenciar as filas de operações que chegam ao DW e decidir qual tipo de operação, consulta ou atualização, deve ter sua execução realizada pelo DW. Após a escolha do tipo de operação que será executada, a transação deste tipo é obtida da fila seguindo um algoritmo *First In, First Out* (FIFO). Esta proposta também utiliza fragmentação do DW buscando criar múltiplas filas de atualização e paralelizar a execução de tais operações.

SHI *et al.* (2009) apresentam um algoritmo para balanceamento das operações de entrada do DW. Nele, as operações são separadas em filas para consultas e atualizações, porém cada operação na fila pode ter níveis de importância diferentes. Assim, inserções de dados considerados prioritários podem ocorrer na frente de outras inserções, o que não ocorre nas soluções que seguem algoritmos FIFO.

MARTINS (2009) propõe uma arquitetura chamada BRAHMA que busca a adequação do ambiente de DW às necessidades dos usuários. Cada usuário tem o seu perfil cadastrado no Módulo de Perfis, indicando um critério de temporalidade necessário para os dados. Com isso o Módulo Gerador de Instâncias ETL cria expressões de extração agrupadas para atender os dados necessários de cada perfil e que são executadas de acordo com o fator de temporalidade de cada perfil. Esta proposta busca separar o processo de extração e inserção dos dados que os usuários necessitam mais atualizados, para que, ao realizar apenas a inserção deles, tais dados sejam disponibilizados mais rapidamente no DW.

Propostas que seguem a linha de ordenação das operações de entrada dependem constantemente de um retorno do usuário sobre como ele deseja que o DW se comporte. Assim, sua implementação pode ser trabalhosa e sujeita a alterações constantes de acordo com novas necessidades dos usuários. Dessa forma, torna-se

mais interessante buscar por alternativas que permitam manter o DW com os dados atualizados sem impactar o tempo de processamento das consultas. Por outro lado, vale considerar que, dado que estas propostas lidam com a problemática do armazenamento de dados em tempo real sem alterar o DW, propostas desta categoria podem ser utilizadas em complemento a outros trabalhos que focam Real-Time DW.

3.2.2. Data Warehouses Virtuais

THO e TJOA (2004) e DOKA *et al.* (2010) propõem a inserção de dados em tempo real diretamente em estruturas de cubos OLAP, permitindo visualizá-los através de ferramentas OLAP como se os mesmos estivessem inseridos em um Data Warehouse.

THO e TJOA (2004) propõem o Grid-based Zero-Latency Data Stream Warehouse (GZLDSWH). Nele, os dados são inseridos em nós de um *grid* computacional no formato em que são obtidos pelas fontes. O GZLDSWH implementa todas as tarefas necessárias sobre estes dados, tais como integração, armazenamento e análise, através de serviços de *grid*. Um destes serviços de grid é o responsável pela construção de um DW virtual com os dados operacionais, fornecendo assim visões de cubos para atendem as necessidades OLAP dos usuários a partir dos dados recém-inseridos.

DOKA *et al.* (2010) apresenta o *Time Series Dwarf*, um sistema de armazenamento de cubos de dados em ambientes *peer-to-peer* (P2P) que permite a atualização dos mesmos em tempo real. Assim, ferramentas OLAP podem disponibilizar dados mais recentes para análise por seus usuários.

O principal problema desta categoria de solução é que outros mecanismos de análises de dados da arquitetura de BI, como *data mining* e *dashboards*, continuam sem acesso aos dados mais recentes, tornando esta solução incompleta para ser considerada como um mecanismo de Real Time Data Warehousing.

3.2.3. Estrutura do Data Warehouse

Outra categoria de proposta para esta área são as que buscam modificar a estrutura do Data Warehouse para que o mesmo suporte inserções continuamente. Propostas desta categoria apresentam como solução a separação entre os dados inseridos em tempo real dos previamente armazenados no DW. Dessa forma, as atualizações de índices e de visões materializadas se tornam menos impactantes, pois os mesmos ou são desabilitados, ou ocorrem em massas menores de dados .

Neste ponto, INMON e STRAUSS (2008) apresentam um trabalho bastante completo onde descrevem diversas inovações que um DW deve tratar a fim de que o

mesmo evolua e seja capaz de atender a diversos requisitos do BI 2.0. Dentre as novas funcionalidades se ressaltam as seguintes:

- Capacidade de realizar inserções online;
- Armazenamento de dados não-estruturados;
- Definição de ciclo de vida para os dados do DW.

Segundo a proposta, para que a execução de cargas contínuas seja possível, deve ser criado um ambiente segregado chamado de setor interativo (*interactive sector*). Neste ambiente, os dados são armazenados conforme obtidos das fontes operacionais, evitando transformações para diminuir os tempos de inserção. Posteriormente, quando se tornar desejável a execução de consultas mais complexas sobre estes dados obtidos em tempo real, os mesmos são integrados, transformados e movidos para um novo ambiente chamado de setor integrado (*integrated sector*). A Figura 14 apresenta cada um dos setores propostos, o ciclo de vida dos dados entre eles e os tipos de dados armazenados. Nesta proposta, são definidos quatro setores (Interativo, Integrado, *Near Line* e de Arquivamento). Os dados que são muito recentes, menos do que 24 horas, são carregados no setor com estrutura igual ao dos sistemas transacionais. Os dados dos outros setores são carregados em estruturas iguais às do DW. Dados não estruturados são sempre carregados em setores com estrutura do DW.

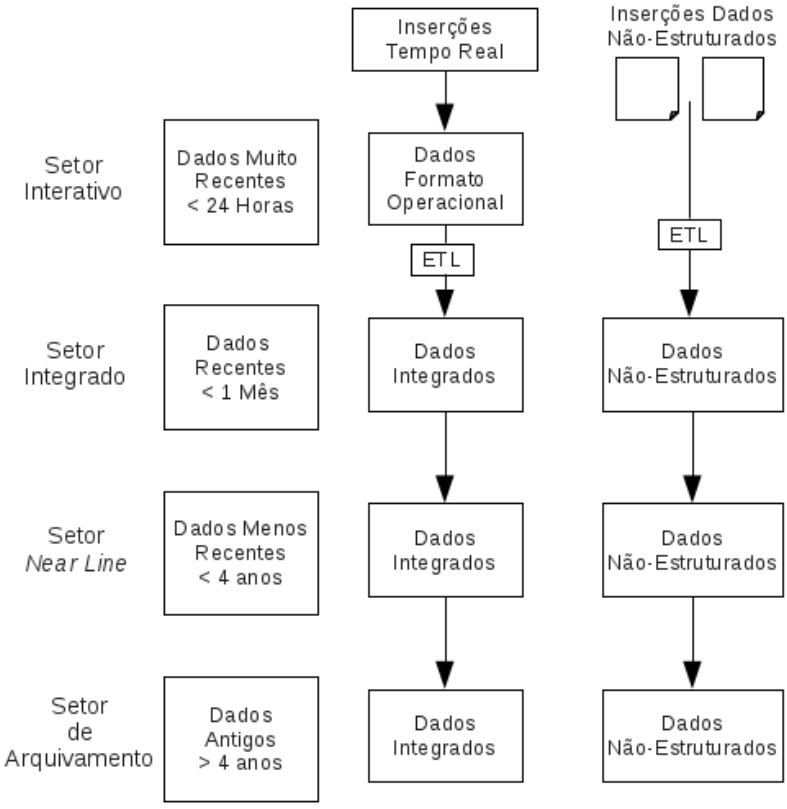


Figura 14 - Modelo de Data Warehouse – adaptado de INMON e STRAUSS (2008)

Entretanto, o armazenamento dos dados com menor latência em formato operacional dificulta a execução de análises mais complexas nos mesmos. Por exemplo, consultas que consideram o setor interativo devem ser elaboradas considerando a estrutura do DW e a estrutura dos dados operacional, usado por este setor. Algumas propostas evoluíram para permitir a inserção em tempo real de dados já transformados, como THOMSEN *et al.* (2008) e SANTOS e BERNARDINO (2009).

THOMSEN *et al*. (2008) propõem como solução um *middleware*, chamado de RiTE, capaz de armazenar em memória os dados a serem inseridos no Data Warehouse, já transformados por ferramentas de Real Time ETL, e realizar consultas nos mesmos. Quando a quantidade de dados atinge determinado limite, é realizado um ETL para carregá-los no DW.

A Figura 15 apresenta tal arquitetura. Nela, ferramentas que coletam dados de fontes operacionais (produtor) utilizam o conector JDBC desenvolvido para inserir no Data Warehouse. Entretanto, tal conector realiza as inserções no banco em memória denominado RiTE *catalyst*. Enquanto isso, usuários (consumidor) acessam o Data Warehouse também através do conector JDBC desenvolvido, o qual encaminha as consultas tanto para o Data Warehouse quanto para o RiTE *catalyst.*

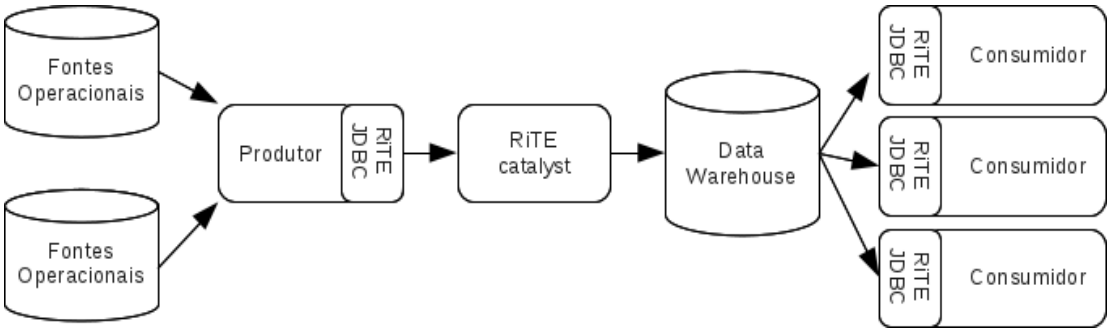


Figura 15 - Arquitetura do RiTE – Adaptado de THOMSEN *et al*. (2008)

Durante a avaliação de desempenho da proposta, o autor apresenta que a realização de consultas nas tabelas que residem no componente RiTE *catalyst* é pior do que consultas em dados no SGBD. Assim, esta solução pode ter um desempenho pior durante a análise dos dados mais recentes.

SANTOS e BERNARDINO (2008) propõem uma adaptação nas tabelas do DW. Novas tabelas devem ser criadas com a mesma estrutura das existentes no DW original, porém sem os mecanismos de otimização de leitura existentes nas tabelas originais, como, por exemplo, sem índices. Estas tabelas são acrescentadas ao modelo tradicional e tornam-se o destino dos dados obtidos das fontes operacionais em tempo real. Quando o desempenho de acesso aos dados armazenados nestas tabelas se torna inaceitável, devido à ausência dos mecanismos de otimização, os dados são extraídos e carregados nas tabelas tradicionais do DW que coexistem com

este modelo novo. A Figura 16 apresenta o modelo de dados proposto. Nela os dados históricos ficam armazenados na tabela fatos Sales e nas dimensões Store e Customer. Já as inserções ocorrem nas tabelas temporárias SalesTmp, StoreTmp e CustomerTmp, as quais, para facilitar a inserção, não possuem regras de integridade habilitadas.

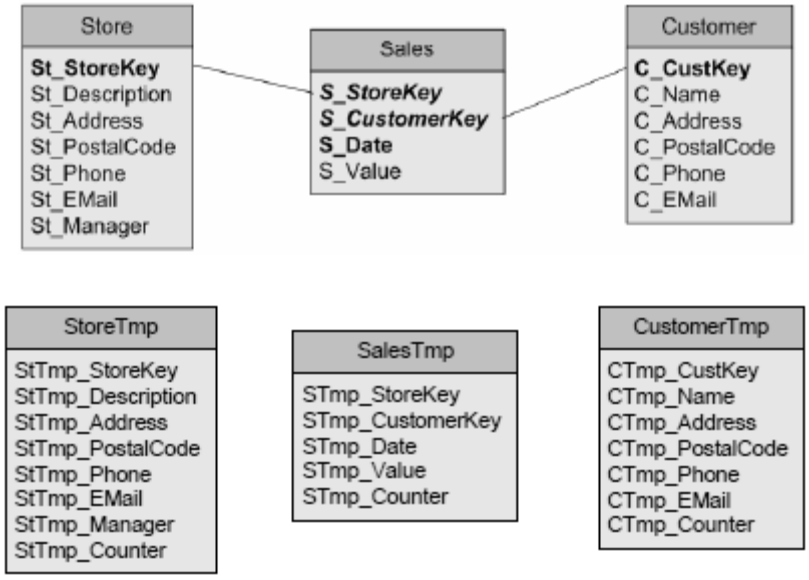


Figura 16 - Data Warehouse proposto por (SANTOS e BERNARDINO, 2008)

Porém, por utilizar objetos distintos para armazenar os dados obtidos em tempo real, as consultas realizadas por ferramentas de análise necessitam ser alteradas para acessar também estas novas tabelas. Esta tarefa gera diversos impactos, como aumento na complexidade das consultas, influência nos planos de execução gerados pelos otimizadores de consultas dos bancos de dados e a reconfiguração de todos os *dashboards*, *scorecards* e cubos existentes em ambientes que já possuem um BI consolidado.

Na arquitetura de BI, o usuário final não acessa diretamente o DW para obter suas informações. Ele faz uso dos mecanismos de *dashboards*, *scorecards*, cubos e relatórios que apresentam uma interface mais amigável e facilitam a obtenção de informações para tomada de decisão. Assim, o conhecimento de localidade dos dados na proposta acima não alcança o usuário final do BI pela própria forma como os dados são tratados para o mesmo. Entretanto, os mecanismos listados acima, os quais são os efetivos usuários do DW, passam a necessitar deste conhecimento, aumentando a complexidade das suas configurações e tornando o trabalho de quem os constrói suscetível a erros.

Uma solução viável para a realização de cargas contínuas no DW requer não apenas um desempenho aceitável durante as mesmas, mas também garantir que os

dados recém inseridos possam ser acessados em conjunto com os dados históricos sem que as ferramentas de análise precisem conhecer sua localidade.

Uma forma de alcançar o isolamento desejado é através do uso de técnicas de distribuição de dados, como um Sistema Gerenciador de Bancos de Dados Distribuídos (SGBDD). Tal mecanismo fornece o conceito de transparência de consultas e atualizações, os quais permitem ao usuário manipular e consultar dados como se os mesmos estivessem em um único repositório, e não fragmentados em múltiplos locais (ÖZSU e VALDURIEZ, 1999) (RISCH, 2009).

3.2.4. Comparações entre as Propostas

As propostas existentes na literatura e apresentadas nesta seção podem ser agrupadas de acordo com a abordagem adotada para solucionar o armazenamento dos dados em tempo real. Cada uma destas abordagens possui vantagens e desvantagens que devem ser avaliadas de acordo com as necessidades do ambiente que as implementa. A Tabela 2 apresenta uma visão comparativa destas propostas de forma resumida.

Tabela 2 – Comparação entre Propostas para Armazenamento de Dados em Tempo Real

Área	Vantagens	Desvantagens
Ordenação	Disponibilizam dados diretamente no DW; Não necessitam efetuar alterações no DW; Atuam em conjunto com outras propostas;	Implementação trabalhosa; Sujeito a constantes modificações;
Virtual	Independem de um DW; Melhor desempenho de ferramentas OLAP sobre os dados;	Ferramentas Não-OLAP não possuem acesso aos dados mais recentes; Não permite análise simultânea de dados históricos;
Estrutura	Disponibilizam dados diretamente no DW; Operações são executadas sem necessidade de ordenação;	Acesso aos dados não ocorre de forma transparente para o usuário;

4. Detalhamento da Solução Proposta

Este trabalho foca em resolver os problemas relacionados ao armazenamento de dados em tempo real em Data Warehouses. Os problemas relacionados às áreas de obtenção e transformação de dados foram abstraídos através do uso do Star Schema Benchmark que foi adaptado para permitir a simulação de inserções em tempo real.

4.1. Cenário de Data Warehouse Tradicional

Tradicionalmente, administradores de bancos de dados do tipo Data Warehouse buscam lidar eficientemente com os *terabytes* de dados que estes bancos armazenam. Seu maior desafio é conseguir permitir o acesso eficiente a tais dados. Para tanto, eles se apóiam em diversos recursos, desde indexação e visões materializadas, disponíveis em SGBDs comuns, até fragmentação e distribuição de dados, a fim de paralelizar a execução das consultas (FURTADO, 2004). A criação eficiente destes recursos depende de uma análise do modelo do Data Warehouse e do comportamento de acesso previsto de seus usuários.

Para lidar com novas inserções de dados nestes bancos gigantes, uma das técnicas utilizadas é acumular grandes quantidades de inserções para execução em lotes, compreendendo grandes intervalos de tempo de transações. Conseqüentemente, estas operações acabam por demandar muito tempo de execução, não só por seu volume, mas também pela quantidade de atualizações de recursos utilizados no Data Warehouse para minimizar os tempos de consultas como, por exemplo, atualização de índices. Assim, para minimizar o tempo destas inserções, geralmente os mecanismos de otimização empregados são desabilitados durante a carga e são atualizados apenas quando esta termina. Dessa forma, há uma indisponibilidade temporária do Data Warehouse, e estas operações de atualização ocorrem normalmente em períodos de baixa utilização do mesmo.

Os recursos utilizados nos Data Warehouses para otimização de leitura possuem funcionalidades distintas e freqüentemente são utilizados em conjunto para obter um melhor desempenho final. Índices permitem acessos mais rápidos a linhas das tabelas, sendo tradicionalmente criados nas colunas mais utilizadas nos filtros das consultas. Sua importância para o Data Warehouse é tamanha que diversos tipos foram desenvolvidos para atender especificamente os problemas deste tipo de banco (JOHNSON, 2009). Já as visões materializadas são criadas especialmente para realizar o pré calculo de agregados, diminuindo o grupo de dados acessados durante consultas que se baseiam nestes agregados e a quantidade de cálculos necessária em operações sobre esses dados.

Com relação à fragmentação e distribuição, FURTADO (2004) apresenta que estas técnicas são tradicionalmente aplicadas em um Data Warehouse da seguinte maneira:

- Fragmenta-se o mais pesado dos elementos de processamento do Data Warehouse, que é a tabela fato, em pedaços menores usando uma distribuição randômica ou *round-robin*.
- As dimensões, muito menores que a tabela fato e tipicamente parte de junções em consultas, são completamente replicadas entre os nós.
- Estruturas auxiliares, como índices ou visões materializadas, são replicadas ou fragmentadas, dependendo da sua associação com os fatos fragmentados ou as dimensões replicadas.

A Figura 17 ilustra como ocorreria o processo de fragmentação em um Data Warehouse composto de uma tabela fatos F e dimensões D. Nela, a tabela de fatos foi fragmentada em F_i pedaços, e as dimensões D foram completamente replicadas para cada fragmento F_i.

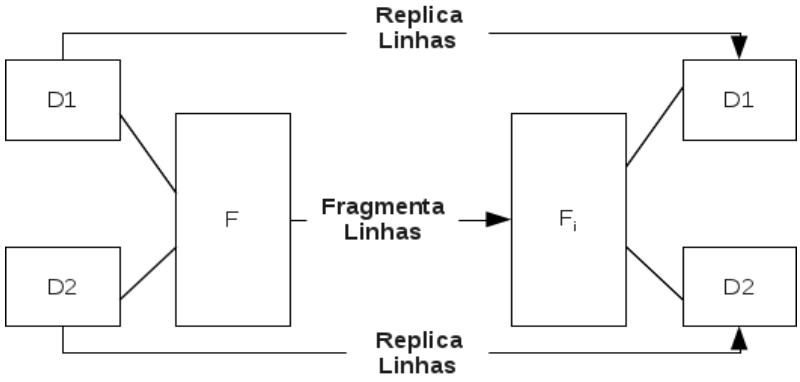


Figura 17 - Fragmentação Tradicional de um Data Warehouse – Adaptado de FURTADO (2004)

Cada um destes F_i fragmentos e as respectivas dimensões D replicadas podem ser então alocados a uma unidade de processamento independente, por exemplo, nós

de um cluster *shared nothing*, como ilustrado na Figura 18. Assim o valor de i varia de 1 a n, onde n é o número de nós do cluster.

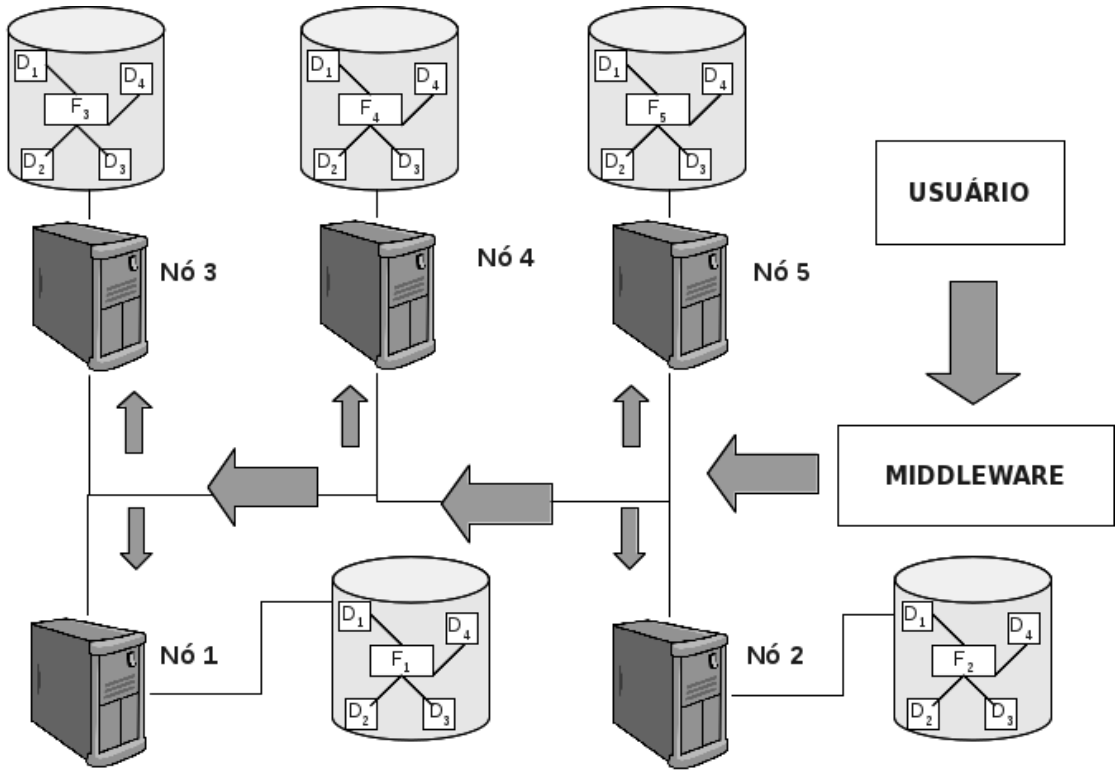


Figura 18 - Arquitetura de Distribuição de Fragmentos Tradicional

O objetivo da proposta de fragmentação de FURTADO (2004) é permitir a quebra de uma consulta em pedaços menores que podem ser executados independentemente, chamado de paralelismo intra-consulta, e assim alcançar um aumento de desempenho bastante próximo ao número de nós adicionados à distribuição. Já a replicação das dimensões tem como objetivo diminuir o conjunto de dados que precisam ser trocados entre os nós durante a execução das sub-consultas.

Entretanto, as técnicas empregadas levam em consideração apenas o desempenho das consultas. Dessa forma, este cenário não é focado na otimização dos tempos das cargas. Com a necessidade da realização de cargas contínuas para permitir a funcionalidade de Real Time Data Warehousing, o comportamento destas inserções de dados se torna componente fundamental no desempenho do DW e das aplicações que se apoiam nos seus dados.

4.2. Arquitetura de Data Warehouse Proposta

Conforme apresentado no Capítulo 3, o cenário de Data Warehouse 2.0 se diferencia do DW tradicional por diversos fatores, entre eles pela necessidade de realização de cargas contínuas. De acordo com a modelagem dimensional, os dados

da tabela fatos estarão relacionados à dimensão temporal em algum campo indicando a data em que o negócio efetivamente ocorreu. O valor deste campo será o mesmo dia em que ele está sendo inserido, já que as cargas contínuas que ocorrem no Data Warehouse 2.0 buscam inserir os dados de fatos recém-ocorridos no negócio. Assim, a dimensão temporal pode ser utilizada como critério de fragmentação a fim de segregar estes dados recém inseridos em um fragmento específico através da técnica de fragmentação horizontal derivada.

Para ilustrar tal abordagem, utilizaremos como base o modelo de dados do Star Schema Benchmark (SSB) (O'NEIL *et al*., 2009). O SSB é um benchmark para Data Warehouses baseado no TPC-H (TPC-H, 2010), com a diferença de que seu modelo de dados segue a linha da modelagem dimensional proposta por KIMBALL (1998). Tal benchmark também é utilizado na execução de testes experimentais da proposta deste trabalho. Detalhes do benchmark são apresentados na Seção 5.1.

A Figura 19 apresenta o modelo de dados do SSB. Nela, a tabela fatos LINEORDER está relacionada a uma dimensão temporal DATE, nos campos ORDERDATE e COMMITDATE. Segundo a descrição deste modelo, o primeiro campo está relacionado a quando o pedido foi realizado e o segundo a quando o pedido foi concluído. Considerando-se que para este negócio os dados são necessários para análise no Data Warehouse após a conclusão do pedido, o valor da coluna COMMITDATE foi escolhido para refletir a data da inserção da linha no Data Warehouse, tendo em vista a realização das cargas contínuas para atender ao cenário de tempo real. Caso os dados fossem inseridos antes da conclusão do pedido a coluna ORDERDATE também deveria ser utilizada como critério para refletir a data de inserção no Data Warehouse.

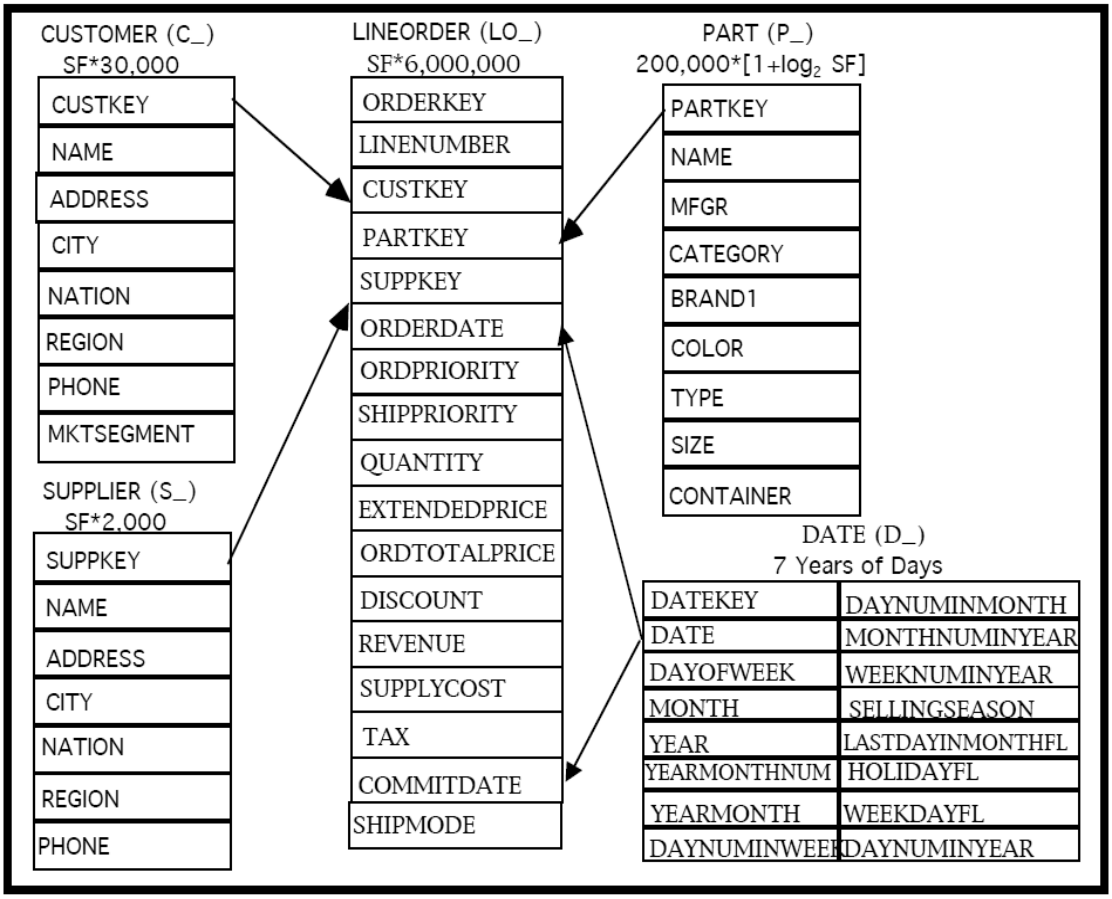


Figura 19 - Modelo de dados do SSB (O'NEIL *et al*., 2009)

Os passos da proposta de fragmentação deste trabalho são apresentados a seguir e estão também publicados em (PEREIRA *et al.*, 2011):

- Identificar o campo da tabela fatos cujo relacionamento com a dimensão temporal indica a data de realização da principal atividade do negócio, importante para as análises.
- Fragmentar a tabela fatos utilizando o campo escolhido, criando um fragmento específico para inserções de eventos recém-ocorridos (chamado de fragmento de tempo real) e outros fragmentos para distribuição dos dados históricos.
- Replicar completamente as dimensões entre os nós, dado que estas são muito menores que a tabela fato e tipicamente fazem parte de junções em consultas.
- Nos fragmentos contendo os dados históricos, replicar ou fragmentar as estruturas auxiliares, como índices ou visões materializadas, de acordo com sua associação com os fatos fragmentados ou as dimensões replicadas. Assim tais estruturas terão informações apenas sobre o subconjunto de dados que o fragmento possui.

- No fragmento de tempo real não criar nenhuma estrutura auxiliar para otimização de consultas, como índices ou visões materializadas.

A Figura 20 ilustra como ocorreria a fragmentação da tabela fatos segundo a proposta, considerando um fragmento de tempo real (Fragmento 1) e quatro fragmentos de dados históricos (Fragmentos 2, 3, 4 e 5). Nela, a data considerada atual é 12/04/2010.

Os dados históricos do Data Warehouse são distribuídos entre os múltiplos fragmentos históricos utilizando-se algum critério definido pelo administrador do banco de dados. A escolha deste critério é bastante flexível e não impactará nas cargas contínuas que ocorrerão em outro fragmento, podendo ser desde um algoritmo *hash*, uma distribuição *round-robin*, ou até uma organização de acordo com a idade dos dados seguindo o princípio defendido por INMON e STRAUSS (2008) para o Data Warehouse 2.0 de que os dados mais antigos de um negócio são menos comumente acessados. O exemplo apresentado na Figura 20 realiza uma distribuição *round-robin*.

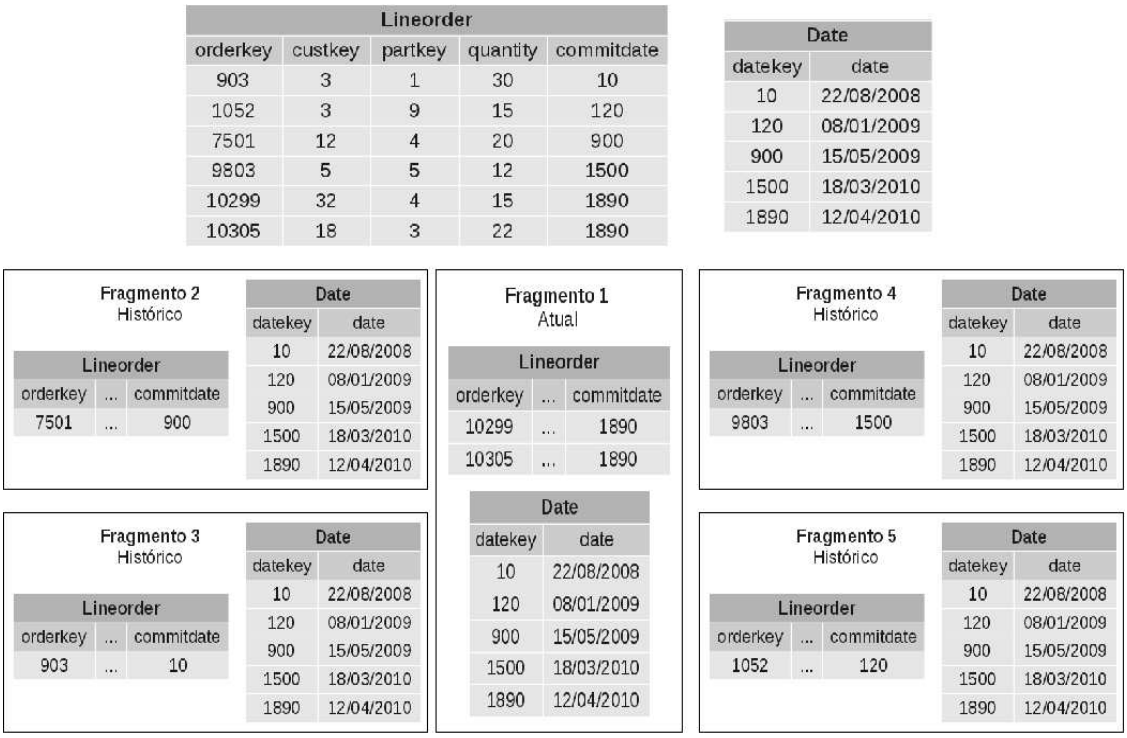


Figura 20 - Arquitetura de Fragmentação Proposta

Durante o dia, todas as inserções serão encaminhadas para o fragmento de tempo real (Fragmento 1) tendo em vista o critério de fragmentação adotado. Assim o Fragmento 1 armazenará todos os registros da tabela LINEORDER cujo campo COMMITDATE, critério de fragmentação escolhido, for igual a 12/04/2010.

Como as dimensões foram completamente replicadas, qualquer inserção que ocorra nelas será encaminhada para todos os nós, inclusive nos que contém fragmentos históricos. Porém, umas das principais características da modelagem

dimensional é o chamado *Slowly Changing Dimensions* (KIMBALL e CASERTA, 2004). Tal conceito descreve que as dimensões não são freqüentemente atualizadas, fazendo com que o custo de novas inserções nelas não seja tão alto para o cenário de cargas contínuas de um Data Warehouse.

Após a etapa de fragmentação dos dados, técnicas de alocação de dados são aplicadas aos fragmentos a fim de associá-los a unidades de processamento distintas, conforme empregado no cenário tradicional. O principal diferencial é que inserções ocorrerão apenas no nó onde o fragmento de tempo real estiver alocado, dado que através da fragmentação o destino das cargas contínuas foi separado.

A Figura 21 ilustra a técnica de distribuição proposta. Nela o nó 1 é o responsável por receber as inserções em tempo real, enquanto os outros nós possuem os dados históricos do DW distribuídos entre si. Um *middleware* recebe as inserções e consultas a serem realizadas no Data Warehouse, encaminhando todas as inserções para o nó 1, enquanto que as consultas são encaminhadas para todos os nós, incluindo o nó de tempo real.

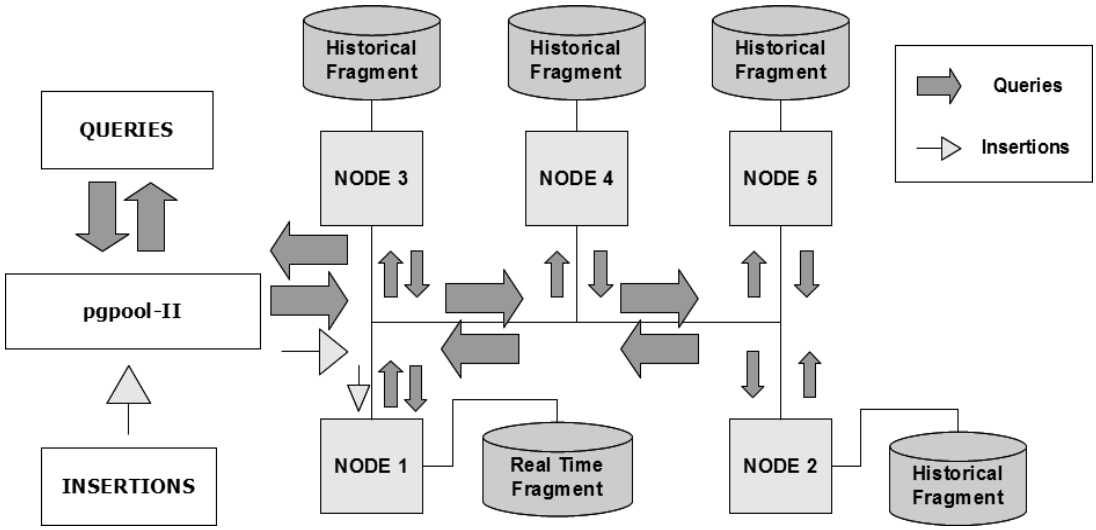


Figura 21 – Exemplo de distribuição dos fragmentos proposta

Na última etapa da carga no cenário tradicional são criados os índices e visões materializadas desejados em cada um dos fragmentos, a fim de otimizar a execução das consultas nos mesmos. Porém, nesta proposta a massa de dados do fragmento de tempo real será substancialmente inferior a dos outros fragmentos, considerando que a quantidade de tuplas existentes no primeiro fragmento corresponderá a apenas um intervalo curto de transações enquanto que nos fragmentos históricos corresponderá a milhares de dias. Assim, todos os mecanismos de otimização de consultas criados não são fundamentalmente necessários para o fragmento de tempo real (nó 1 da Figura 21). Ao contrário, eles acabariam por prejudicar as execuções constantes de inserções neste fragmento, pois, após cada inserção, tais mecanismos

necessitariam ser atualizados com a nova informação, aumentando o consumo de processamento da máquina e diminuindo o tempo de resposta do SGBD.

Portanto, neste trabalho, é considerado que os mecanismos de otimização amplamente utilizados em Data Warehouses tradicionais devem ser desabilitados exclusivamente no fragmento de tempo real, permitindo um tempo de inserção mais rápido.

Por outro lado, com o aumento constante da quantidade de dados no fragmento de tempo real, é necessário avaliar continuamente o seu tamanho, a fim de avaliar o impacto no tempo da realização de consultas que o incluam. A proposta apresentada por SANTOS e BERNARDINO (2009) de Data Warehouse também enfrenta problema parecido durante suas inserções. Para tanto os autores apresentam os passos da Figura 22 para tratar este fato.

```
Adaptar o Data Warehouse para suportar cargas contínuas;
Realizar cargas contínuas;
Se o desempenho de aplicações OLAP for aceitável:
    Continuar realizando cargas;
Senão:
    Agrupar dados recentes;
    Inserir estes dados como históricos;
```

Figura 22 – Redistribuição de dados recentes – Adaptado de SANTOS e BERNARDINO (2009)

Apesar de identificarem a necessidade de redistribuir os dados, os autores informam que tal atividade não faz parte do escopo de sua pesquisa e, portanto, não aprofundam na mesma. Eles indicam apenas que os critérios para definir quando ocorreria tal distribuição poderiam ser: (i) o número de registros inseridos nas tabelas temporárias; (ii) a quantidade de espaço físico ocupado pelas mesmas; (iii) um período pré-definido de tempo; ou, até mesmo, (iv) uma intervenção manual do administrador do banco de dados.

Considerando a escolha de um critério para testar o desempenho do nó de tempo real, a definição de um limite para este critério e a definição de um período de tempo para realização do teste, o algoritmo para realizar a redistribuição pode ser descrito como apresentado na Figura 23.

```
A cada período de tempo para realização de teste
Calcular valor do teste de acordo com o critério escolhido;
Se Valor de teste > Valor Limite Definido:
    Redistribuir dados do fragmento de tempo real para os históricos;
```

Figura 23 - Algoritmo para redistribuir dados do fragmento de tempo real

A escolha do critério a ser adotado, assim como dos seus valores limites, dependerá de análises prévias do comportamento do fragmento com o crescimento do seu volume de dados. Porém, um possível critério a ser adotado a fim de evitar tal análise seria a definição de consultas que, enquanto tiverem um tempo de resposta aceitável, indicariam a não necessidade de redistribuir os dados. Tempo de resposta aceitável pode ser definido como um tempo de execução da consulta no nó de tempo real menor do que o da execução da mesma consulta nos fragmentos históricos. Dessa forma, a parte da consulta executada no fragmento de tempo real não seria a responsável pelo atraso do desempenho da consulta original, não afetando o desempenho das aplicações OLAP. Assim, o algoritmo ficaria:

```
A cada período de tempo para realização de teste (ex: 10 minutos)
Executar consulta Q1;
Se Tempo Q1 > Tempo Q1 medido no DW:
    Executar ETL para redistribuir dados do fragmento de tempo real para os fragmento históricos;
```

Figura 24 – Algoritmo proposto para redistribuição do fragmento de tempo real

5. Testes Experimentais

Para poder validar a proposta deste trabalho foram realizados experimentos, os quais são descritos neste capítulo. Tais experimentos foram executados com base em um *benchmark* para Data Warehouses, amplamente utilizado na literatura, o Star Schema Benchmark (O'NEIL *et al*., 2009).

5.1. Star Schema Benchmark

O *benchmark* denominado Star Schema Benchmark (SSB) (O'NEIL *et al*., 2009) é uma adaptação do *benchmark* de sistemas de suporte a decisão TPC-H (TPC-H, 2010) onde o modelo do Data Warehouse é convertido para o modelo estrela e suas consultas são adaptadas. Seu modelo de dados foi apresentado na Figura 19 (Seção 4.2) durante a descrição da arquitetura proposta por este trabalho. Este *benchmark* é composto de treze consultas agrupadas em quatro categorias.

A primeira categoria (Q1) é composta por três consultas, caracterizadas por realizarem restrições em apenas uma dimensão: temporal (date). Tais consultas buscam quantificar o aumento de rendimento que ocorreria no negócio através da eliminação de alguns percentuais de desconto em um dado grupo de produtos enviados em um determinado ano. A Figura 25, a Figura 26 e a Figura 27 apresentam estas consultas.

```
select sum(lo_extendedprice*lo_discount) as revenue
from lineorder, date
where lo_orderdate = d_datekey
    and d_year = 1993
    and lo_discount between 1 and 3
    and lo_quantity< 25
```

Figura 25 – Consulta Q1.1 do Star Schema Benchmark (O'NEIL *et al*., 2009)

```
select sum(lo_extendedprice*lo_discount) as revenue
from lineorder, date
where lo_orderdate = d_datekey
    and d_yearmonthnum = 199401
    and lo_discount between 4 and 6
    and lo_quantity between 26 and 35
```

Figura 26 – Consulta Q1.2 do Star Schema Benchmark (O'NEIL *et al*., 2009)

```
select sum(lo_extendedprice*lo_discount) as revenue
from lineorder, date
where lo_orderdate = d_datekey
    and d_weeknuminyear = 6
    and d_year = 1994
    and lo_discount between 5 and 7
    and lo_quantity between 26 and 35
```

Figura 27 – Consulta Q1.3 do Star Schema Benchmark (O'NEIL *et al*., 2009)

A segunda categoria (Q2) é composta por três consultas, caracterizadas por realizarem restrições em duas dimensões: produto (part) e fornecedor (supplier). Tais consultas buscam comparar a rentabilidade por algumas classes de produtos, por fornecedores de uma determinada região, agrupando por marcas de produtos e o ano em que a transação foi realizada. A Figura 28, a Figura 29 e a Figura 30 apresentam estas consultas.

```
select sum(lo_revenue), d_year, p_brand1
from lineorder, date, part, supplier
where lo_orderdate = d_datekey
    and lo_partkey = p_partkey
    and lo_suppkey = s_suppkey
    and p_category = 'MFGR#12'
    and s_region = 'AMERICA'
group by d_year, p_brand1
order by d_year, p_brand1
```

Figura 28 – Consulta Q2.1 do Star Schema Benchmark (O'NEIL *et al*., 2009)

```
select sum(lo_revenue), d_year, p_brand1
from lineorder, date, part, supplier
where lo_orderdate = d_datekey
    and lo_partkey = p_partkey
    and lo_suppkey = s_suppkey
    and p_brand1 between 'MFGR#2221' and 'MFGR#2228'
    and s_region = 'ASIA'
group by d_year, p_brand1
order by d_year, p_brand1
```

Figura 29 – Consulta Q2.2 do Star Schema Benchmark (O'NEIL *et al*., 2009)

```
select sum(lo_revenue), d_year, p_brand1
from lineorder, date, part, supplier
where lo_orderdate = d_datekey
    and lo_partkey = p_partkey
    and lo_suppkey = s_suppkey
    and p_brand1= 'MFGR#2239'
    and s_region = 'EUROPE'
group by d_year, p_brand1
order by d_year, p_brand1
```

Figura 30 – Consulta Q2.3 do Star Schema Benchmark (O'NEIL *et al*., 2009)

A terceira categoria (Q3) é composta por quatro consultas, caracterizadas por realizarem restrições em três dimensões: temporal (date), fornecedor (supplier) e cliente (customer). Tais consultas buscam fornecer o volume de rendimento para as transações por país do cliente, país do fornecedor e ano da transação, dado certa região e um período de tempo. A Figura 31, a Figura 32, a Figura 33 e a Figura 34 apresentam estas consultas.

```
select c_nation, s_nation, d_year, sum(lo_revenue) as
revenue
from customer, lineorder, supplier, date
where lo_custkey = c_custkey
    and lo_suppkey = s_suppkey
    and lo_orderdate = d_datekey
    and c_region = 'ASIA'
    and s_region = 'ASIA'
    and d_year>= 1992 and d_year<= 1997
group by c_nation, s_nation, d_year
order by d_year asc, sum(lo_revenue) desc
```

Figura 31 – Consulta Q3.1 do Star Schema Benchmark (O'NEIL *et al*., 2009)

```
select c_city, s_city, d_year, sum(lo_revenue) as revenue
from customer, lineorder, supplier, date
where lo_custkey = c_custkey
    and lo_suppkey = s_suppkey
    and lo_orderdate = d_datekey
    and c_nation = 'UNITED STATES'
    and s_nation = 'UNITED STATES'
    and d_year>= 1992 and d_year<= 1997
group by c_city, s_city, d_year
order by d_year asc, sum(lo_revenue) desc
```

Figura 32 – Consulta Q3.2 do Star Schema Benchmark (O'NEIL *et al*., 2009)

```
select c_city, s_city, d_year, sum(lo_revenue) as revenue
from customer, lineorder, supplier, date
where lo_custkey = c_custkey
    and lo_suppkey = s_suppkey
    and lo_orderdate = d_datekey
    and (c_city='UNITED KI1' or c_city='UNITED KI5')
    and (s_city='UNITED KI1' or s_city='UNITED KI5')
    and d_year>= 1992 and d_year<= 1997
group by c_city, s_city, d_year
order by d_year asc, sum(lo_revenue) desc
```

Figura 33 – Consulta Q3.3 do Star Schema Benchmark (O'NEIL *et al*., 2009)

```
select c_city, s_city, d_year, sum(lo_revenue) as revenue
from customer, lineorder, supplier, date
where lo_custkey = c_custkey
    and lo_suppkey = s_suppkey
    and lo_orderdate = d_datekey
    and (c_city='UNITED KI1' or c_city='UNITED KI5')
    and (s_city='UNITED KI1' or s_city='UNITED KI5')
    and d_yearmonth = 'Dec1997'
group by c_city, s_city, d_year
order by d_year asc, sum(lo_revenue) desc;
```

Figura 34 – Consulta Q3.4 do Star Schema Benchmark (O'NEIL *et al*., 2009)

A quarta categoria (Q4) é composta por três consultas. Tais consultas buscam representar uma seqüência OLAP do tipo "*What-If*", aonde novas condições vão sendo elaboradas a fim de estudar impactos em uma determinada medida. Neste caso, a medida avaliada é o lucro acumulado das transações (lo_revenue – lo_supplycost). A Figura 35, a Figura 36 e a Figura 37 apresentam estas consultas.

```
select d_year, c_nation, sum(lo_revenue - lo_supplycost)
as profit
from date, customer, supplier, part, lineorder
where lo_custkey = c_custkey
    and lo_suppkey = s_suppkey
    and lo_partkey = p_partkey
    and lo_orderdate = d_datekey
    and c_region = 'AMERICA'
    and s_region = 'AMERICA'
    and (p_mfgr = 'MFGR#1' or p_mfgr = 'MFGR#2')
group by d_year, c_nation
order by d_year, c_nation
```

Figura 35 – Consulta Q4.1 do Star Schema Benchmark (O'NEIL *et al*., 2009)

```
select d_year, s_nation, p_category, sum(lo_revenue -
lo_supplycost) as profit   from date, customer, supplier,
part, lineorder
where lo_custkey = c_custkey
    and lo_suppkey = s_suppkey
    and lo_partkey = p_partkey
    and lo_orderdate = d_datekey
    and c_region = 'AMERICA'
    and s_region = 'AMERICA'
    and (d_year = 1997 or d_year = 1998)
    and (p_mfgr = 'MFGR#1' or p_mfgr = 'MFGR#2')
group by d_year, s_nation, p_category
order by d_year, s_nation, p_category
```

Figura 36 – Consulta Q4.2 do Star Schema Benchmark (O'NEIL *et al*., 2009)

```
select d_year, s_city, p_brand1, sum(lo_revenue -
lo_supplycost) as profit
from date, customer, supplier, part, lineorder
where lo_custkey = c_custkey
    and lo_suppkey = s_suppkey
    and lo_partkey = p_partkey
    and lo_orderdate = d_datekey
    and c_region = 'AMERICA'
    and s_nation = 'UNITED STATES'
    and (d_year = 1997 or d_year = 1998)
    and p_category = 'MFGR#14'
group by d_year, s_city, p_brand1
order by d_year, s_city, p_brand1
```

Figura 37 – Consulta Q4.3 do Star Schema Benchmark (O'NEIL *et al*., 2009)

Por fim, o SSB também dispõe de um gerador de dados chamado DBGEN, adaptado do TPC, que permite a obtenção de uma massa de dados para o seu modelo. Tal massa de dados pode ser gerada em escalas diferentes em múltiplos de 10, através de um parâmetro denominado *Scale Factor*. O menor valor do *Scale Factor* é 1, gerando um banco com aproximadamente 1 GB de dados e uma tabela fatos com seis milhões de tuplas.

O SSB foi desenvolvido para analisar o comportamento de um Data Warehouse tradicional construído a partir da modelagem dimensional. Logo, para que possa ser utilizado na análise do cenário de Real Time Data Warehousing, o SSB necessita de alterações as quais são propostas neste trabalho.

5.2. Real Time Star Schema Benchmark

Real Time Data Warehouses se diferem dos Data Warehouses Tradicionais em diversos aspectos, conforme apresentado no Capítulo 3. Logo, um *benchmark* que permita a análise do comportamento destes DWs necessita simular seus principais pontos, como a execução de cargas contínuas e de consultas nos dados obtidos mais recentemente. A fim de atender tais necessidades, este trabalho realiza algumas alterações no *benchmark* SSB para que o mesmo também possa avaliar o comportamento de Real Time Data Warehouses.

Inicialmente, foi acrescentada uma nova data na dimensão temporal (date) para refletir o dia atual do negócio, no qual as transações estão ocorrendo e sendo inseridas continuamente no Data Warehouse. Como o modelo SSB gera fatos utilizando as datas entre 01/01/1992 e 31/12/1998, foi escolhida a data 01/01/1999 para simbolizar o dia atual. Por conseqüência, o utilitário DBGEN, responsável por gerar a massa de dados, teve que ser alterado para incluir esta data na massa de dados da dimensão temporal.

O DBGEN também foi alterado para gerar massas de dados que refletem as novas transações do negócio (REAL-TIME-SSB, 2011). Estes dados são criados em arquivos separados, a fim de diferenciar a carga de dados histórica da simulação de cargas contínuas. Cada nova transação é representada por uma tupla da tabela fatos (lineorder) onde o valor da coluna lo_commitdate é fixado para 01/01/1999. Esta coluna é quem reflete a data de conclusão da transação de acordo com o modelo do SSB.

Os arquivos para cargas contínuas são gerados com uma quantidade de transações correspondente a 0,1% do total de tuplas históricas da tabela fatos. Este percentual foi definido a partir das especificações do *benchmark* TPC-H para testes de atualizações em Data Warehouses (TPCH, 2010). Assim, como o TPC-H é um *benchmark* bastante difundido no mercado, os tempos das cargas podem ser comparados com os de soluções tradicionais já avaliadas com a métrica do TPC-H.

A massa de dados relacionada a este valor de 0,1% representa algo próximo a quantidades de transações de um dia. Portanto tais dados são suficientes para permitir simulações de acordo com a estratégia de Cargas em Tempo Real adotada pelo usuário.

O tempo de execução de uma carga completa permite verificar o desempenho do Data Warehouse com relação à execução de cargas contínuas. Todavia, a latência da disponibilidade dos dados no Data Warehouse não pode ser avaliada por este valor, já que ela está intimamente ligada à estratégia de Carga em Tempo Real adotada, conforme apresentado na Seção 3.1.

Também foi definida uma nova categoria de consultas para execução em conjunto com as já propostas pelo SSB. O objetivo destas consultas foi avaliar o desempenho no acesso exclusivo aos dados obtidos em tempo real, simulando consultas que verifiquem o estado atual do negócio, um dos objetivos da implantação de um Real Time Data Warehouse. Esta categoria é designada como Q5, em complemento aos já existentes no SSB, e é composta por quatro consultas, derivadas das consultas 1.1, 2.1, 3.1 e 4.1 do SSB com filtros para acesso exclusivo aos dados em tempo real, ou seja, um filtro para acessar apenas dados do dia 01/01/1999.

A escolha das consultas utilizadas para a nova categoria Q5 teve o objetivo de refletir cada uma das categorias criadas pelo SSB. Estas categorias são utilizadas pelo SSB para reunir consultas de acordo com o tipo de operação realizada. Dessa forma, ao adaptarmos ao menos uma consulta de cada categoria, todas as simulações de acesso previstas no SSB também são realizadas objetivando apenas os dados inseridos recentemente na tabela fatos. A Figura 38, a Figura 39, a Figura 40 e a Figura 41 apresentam as consultas que compõem a categoria Q5.

```
select sum(lo_extendedprice*lo_discount) as revenue
from lineorder, date
where lo_orderdate = d_datekey
    and d_date = “January 01, 1999”
    and lo_discount between 1 and 3
    and lo_quantity < 25;
```

Figura 38 – Consulta para acesso a dados em tempo real Q5.1

```
select sum(lo_revenue), d_year, p_brand1
from lineorder, date, part, supplier
where lo_orderdate = d_datekey
    and lo_partkey = p_partkey
    and lo_suppkey = s_suppkey
    and d_date = “January 01, 1999”
    and p_category = 'MFGR#12'
    and s_region = 'AMERICA'
group by d_year, p_brand1
order by d_year, p_brand1;
```

Figura 39 – Consulta para acesso a dados em tempo real Q5.2

```
select c_nation, s_nation, d_year, sum(lo_revenue) as
revenue
from customer, lineorder, supplier, date
where lo_custkey = c_custkey
    and lo_suppkey = s_suppkey
    and lo_orderdate = d_datekey
    and c_region = 'ASIA'
    and s_region = 'ASIA'
    and d_date = “January 01, 1999”
group by c_nation, s_nation, d_year
order by d_year asc, revenue desc;
```

Figura 40 – Consulta para acesso a dados em tempo real Q5.3

```
select d_year, c_nation, sum(lo_revenue - lo_supplycost)
as profit
from date, customer, supplier, part, lineorder
where lo_custkey = c_custkey
    and lo_suppkey = s_suppkey
    and lo_partkey = p_partkey
    and lo_orderdate = d_datekey
    and d_date = “January 01, 1999”
    and c_region = 'AMERICA'
    and s_region = 'AMERICA'
    and (p_mfgr = 'MFGR#1' or p_mfgr = 'MFGR#2')
group by d_year, c_nation
order by d_year, c_nation
```

Figura 41 – Consulta para acesso a dados em tempo real Q5.4

5.3. Ambiente de Testes

Todos os testes foram realizados no laboratório do projeto CGOLAP (CGOLAP, 2009), em um cluster do tipo *shared nothing*, cuja arquitetura é apresentada na Seção 2.1, composto por 6 computadores. Destes computadores, cinco foram utilizados como nós de processamento e estão equipados com processador Intel Core 2 Duo E6750

(2.66 GHz) e 2Gb de memória RAM, rodando o banco de dados PostgreSQL 8.4. O sexto computador é utilizado como nó de entrada e roda o *middleware* pgpool-II versão 2.3.3, responsável pela gerência distribuída dos SGBDs.

O PostgreSQL foi escolhido para os testes por ser um SGBD de código aberto, gratuito e bastante robusto. O mesmo possui desenvolvimento contínuo há mais de quinze anos e vem sendo cada vez mais adotado no mercado, concorrendo com SGBDs comerciais como Oracle e Microsoft SQL Server (POSTGRESQL, 2011).

Com a definição do SGBD, foi necessário escolher um *middleware* compatível com o PostgreSQL e que permitisse construir a arquitetura proposta neste trabalho de um cluster *shared nothing* com dados fragmentados entre os nós.

A Tabela 3 é o resultado do levantamento feito na literatura com relação a soluções existentes para clusterização e replicação no PostgreSQL. Nela, diversos *middlewares* são classificados de acordo com as arquiteturas que suportam. Cenários de clusterização envolvem suporte a arquiteturas *shared nothing, shared disk, shared memory* ou *shared everything* e cenários de replicação envolvem arquiteturas *multi-master* ou *master/slave,* podendo tal replicação ocorrer de forma síncrona ou assíncrona.

Nas replicações do tipo *master/slave*, apenas a réplica considerada como *master*, ou ativa, pode receber operações de escrita, sendo ela responsável por atualizar as réplicas *slaves*, ou passivas, com os dados inseridos. Existem duas variações deste cenário, em um deles as réplicas *slave* não podem ser acessadas até uma delas se tornar ativa, e em outro, chamado de *hot*, as réplicas *slaves* podem ser acessadas para operações de leitura. Já nas replicações do tipo *multi-master*, operações de leitura e escrita podem ser realizadas em quaisquer das réplicas (JIMÉNEZ-PERIS e PATIÑO-MARTÍNEZ, 2009).

Uma replicação ocorrer de forma síncrona significa que todas as suas réplicas são atualizadas simultaneamente, possuindo sempre os mesmos dados. Já uma replicação ocorrer de forma assíncrona significa que as suas réplicas são sincronizadas posteriormente a uma alteração de dados já ter sido concluída em uma delas (WADA, 2009).

Tabela 3 – Soluções de clusterização e replicação para PostgreSQL

Solução	Shared Nothing	Shared Disk	Shared Memory	Shared Everyth.	Multi Master	Master Slave	Sinc.	Assinc.
Slony-I						X		X
Bucardo					X	X		X
Londiste						X		X
pgpool-II	X				X		X	
Tungsten	X				X	X	X	
Postgres -R					X		X	
pg_ standby						X		X
ParGRES	X				X		X	
ParGRES -SmaQSS	X							

Os resultados da tabela reduzem as possibilidades para esta arquitetura em quatro *middlewares*: pgpool-II, ParGRES, Tungsten e ParGRES-SmaQSS.

O ParGRES (MATTOSO *et al*., 2006) é um *middleware* que permite a realização de paralelismo intra-consulta em qualquer SGBD que suporte o padrão SQL. Seu paralelismo busca aperfeiçoar o desempenho de consultas pesadas típicas de aplicações OLAP, decompondo-as em sub-consultas que são executadas em paralelo.

A arquitetura do ParGRES é apresentada na Figura 40. Ela é composta pelo Processador de Consultas de Cluster (*Cluster Query Processor* - CQP) que controla a execução das requisições, pelo Intermediador que permite a comunicação entre as aplicações clientes e o CQP, e pelos Processadores de Consultas de Nó (*Node Query Processor* - NQP) que são responsáveis pela execução das requisições do CQP em cada nó.

Entretanto, apesar do ParGRES permitir a construção de um cluster *shared nothing*, o mesmo trabalha com replicação completa dos dados em cada um dos nós. Assim, o ParGRES não permite a construção do cenário de fragmentação e distribuição de dados proposto neste trabalho.

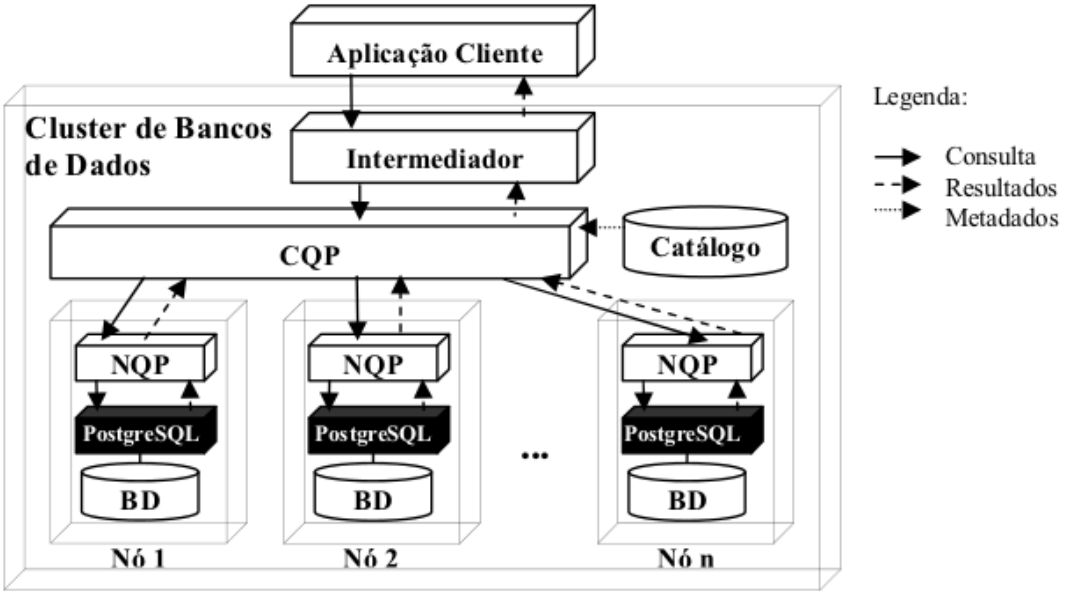


Figura 42 – Arquitetura do ParGRES (MATTOSO *et al.*, 2006)

O Tungsten (TUNGSTEN, 2010) é o novo projeto desenvolvido e suportado pela Continuent, apresentado como alternativa para o Sequoia, antigo C-JDBC, que não será mais mantido pela mesma. Esta solução é dividida em diversos componentes que se integram para ser uma solução completa de replicação e paralelização inter-consulta. O Tungsten permite replicações do tipo *master-slave* e *multi-master*, além disso o mesmo também disponibiliza um JDBC wrapper que é responsável por realizar o balanceamento das consultas e uma recuperação de falhas transparente.

Porém, assim como o ParGRES, o Tungsten também trabalha com replicação completa dos dados em cada um dos nós, não permitindo a construção do cenário de fragmentação e distribuição de dados proposto neste trabalho.

O ParGRES-SmaQSS (PAES *et al.*, 2009) é uma extensão do ParGRES que incorpora o conceito de fragmentação física e virtual no mesmo, se apoiando nos conceitos do protótipo SmaQSS (LIMA *et al*., 2009). Esta solução trabalha com replicação parcial dos fragmentos entre os nós, de forma a evitar que falhas em um determinado nó causem a indisponibilidade de dados.

Por permitir a fragmentação física, o ParGRES-SmaQSS permite a construção da arquitetura definida nesta proposta. Entretanto, a replicação parcial dos fragmentos, que fornece um nível de tolerância a falhas aos clusters que o utilizam, impacta diretamente no desempenho das atualizações (LIMA *et al.*, 2009). Como o principal objetivo desta proposta é o de permitir a execução de cargas contínuas nos Data Warehouses, a utilização deste *middleware* poderia impactar no desempenho desejado.

O pgpool-II (PGPOOL-II, 2009) é um *middleware* criado para existir entre as conexões dos clientes e servidores PostgreSQL. Ele possui diversas funcionalidades,

tais como: pooling de conexões, replicação, balanceamento de carga, paralelização de consultas, fragmentação e distribuição de dados. Esta última funcionalidade é essencial para a configuração de uma arquitetura *shared nothing* com dados fragmentados e distribuídos entre os nós como a proposta por este trabalho.

Por conseguir utilizar diretamente o protocolo de comunicação de entrada e saída do PostgreSQL, as aplicações acessando os dados via pgpool-II acreditam que estão se comunicando diretamente com o PostgreSQL e o PostgreSQL ao receber a conexão acredita que a mesma está sendo originada de um cliente normal. Graças a essa transparência fornecida ao cliente e servidor, aplicações que acessam bancos de dados podem utilizar o pgpool-II com nenhuma alteração em seu código fonte.

O pgpool-II trabalha em quatro modos que oferecem funcionalidades distintas: *Raw*, *Replication*, *Master*/*Slave*, *Parallel Query*. A Tabela 4 apresenta uma comparação entre cada um destes modos.

O modo *Raw* consiste em utilizar o pgpool-II apenas como um redirecionador de conexões para o PostgreSQL. Assim é possível realizar *pooling* de conexões através do mesmo, limitando o número de conexões abertas no PostgreSQL e evitando aberturas e fechamentos constantes de conexões. Além disso, ele também permite realizar operações de *failover* entre múltiplos servidores.

No modo *Replication* o pgpool-II realiza a replicação completa das transações efetuadas no banco em todos os nós cadastrados. Assim, esta configuração permite um ambiente tolerante a falhas enquanto ao menos um nó continuar funcionando normalmente. O *middleware* também realiza um paralelismo inter-consultas, distribuindo as consultas recebidas entre os nós.

O modo *Master*/*Slave* é utilizado para associar o pgpool-II com outros softwares de replicação do tipo *Master*/*Slave*. Assim, o pgpool-II realiza o redirecionamento das conexões em caso de falha do nó *Master*, além de permitir a realização de paralelismo inter-consultas com os nós *Slaves.*

O modo *Parallel Query* é o que suporta a distribuição de dados em uma arquitetura *shared nothing*, conforme a Figura 43 demonstra. Nele o pgpool-II, através de regras cadastradas no seu dicionário de dados, identifica onde operações de inserção/atualização devem se executadas e as encaminha. Assim, este modo permite distribuir as inserções realizadas, além de realizar paralelismo intra-consultas, já que as consultas recebidas acessam fragmentos dos dados ao serem encaminhadas aos nós. O *middleware* é o responsável por unir os resultados parciais fornecidos pelos nós, compondo o resultado final.

Tabela 4 – Modos e Funcionalidades do pgpool-II – Adaptado de (PGPOOL-II, 2011)

Função / Modo	Raw	Replication	Master/Slave	Parallel Query
Pooling de Conexões	X	X	X	X
Replicação		X		X
Balanceamento de Carga		X	X	X
Failover	X	X	X	
Recuperação Online		X	X	
Fragmentação e Distribuição				X

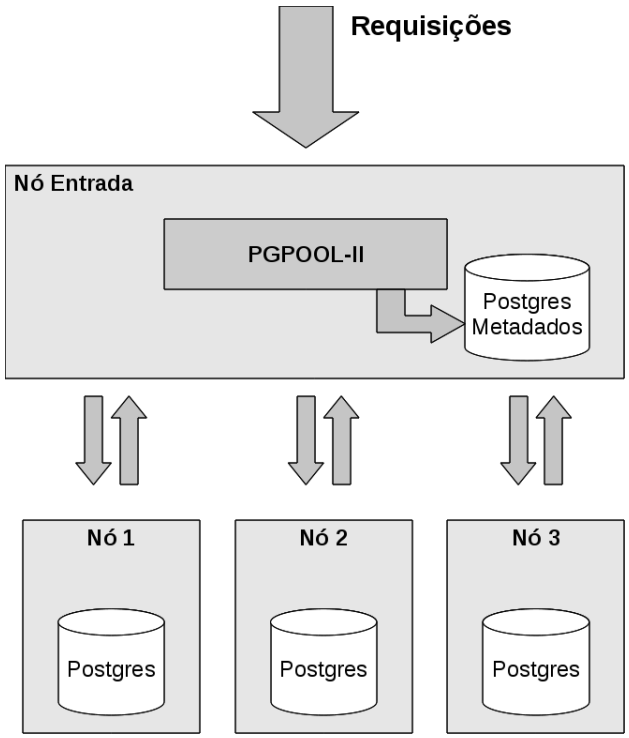


Figura 43 - Arquitetura Parallel Query pgpool-II

Finalizando a descrição das máquinas do ambiente de testes, todos os computadores executam o sistema operacional OpenSUSE 10.3 e estão interconectados através de rede GigaBit. Todos os dados são acessados em um disco exclusivo para cada computador sem configurações de RAID.

O método de carga escolhido para a realização dos testes no Data Warehouse foi Micro Batch ETL. Sua utilização permite o mesmo nível de transformação e integração de dados que ferramentas ETL normais, o que fornece a capacidade de se realizar análises nos dados do DW tão complexas quantos em DW tradicionais. Além disso, sua semelhança com ferramentas de ETL tradicionais durante a construção dos mapas simplifica a sua implementação.

Prosseguindo na preparação do ambiente, foi necessário gerar uma massa de dados para o benchmark através do utilitário dbgen. Para o nosso ambiente, utilizamos um *Scale Factor* de 100, obtendo-se uma tabela fatos com seiscentos milhões de registros.

Após a realização das cargas, em cada um dos cenários, foram habilitadas as regras de integridade previstas no SSB, sendo que as únicas indexações existentes estão relacionadas às chaves primárias das tabelas. A única exceção é o Fragmento de Tempo Real, o qual não possui indexações nem regras de integridade habilitadas, como proposto nesta arquitetura.

5.4. Testes realizados

Os testes foram realizados nos dois cenários de fragmentação apresentados no Capítulo 4:

- Fragmentação de dados utilizada em Data Warehouses tradicionais (FURTADO, 2004).
- Fragmentação de dados para atender à Data Warehouses Real Time (proposta deste trabalho).

No primeiro cenário, os dados são fragmentados através de uma distribuição *round-robin*. Desta forma, os dados são distribuídos de forma balanceada entre os fragmentos, fazendo com que cada nó tenha aproximadamente 20% dos dados totais. A inserção de dados em tempo real neste cenário também ocorre de forma distribuída, com cada nó recebendo uma fração das cargas.

No segundo cenário, quatro dos cinco nós de processamento são definidos como nós de dados históricos e neles são inseridos todos os dados correspondentes à carga inicial através da mesma distribuição *round-robin*. As inserções em tempo real ocorrem no quinto nó, definido, conforme a proposta, como fragmento de tempo real.

As consultas dos procedimentos de comparação foram executadas a frio, seguindo o roteiro abaixo em cada cenário:

- Bancos carregados inicialmente com os dados gerados do DBGEN para o modelo do SSB de acordo com o cenário de fragmentação analisado.
- Flush das áreas de memória dos bancos
- Execução das consultas sem a concorrência de cargas.
- Flush das áreas de memória dos bancos
- Início da carga com um dos arquivos de dados adicionais gerados através da modificação efetuada no DBGEN

- Execução das consultas simultaneamente às cargas.

Em complemento à análise obtida com a execução das consultas também foram realizados os seguintes testes, a fim de validar alguns pontos específicos: (i) análise de deterioração do fragmento de tempo real e (ii) análise do tempo de redistribuição dos dados.

A **análise de deterioração do fragmento de tempo real** corresponde aos testes de deterioração do tempo de consultas e de carga no fragmento de tempo real a medida que seu volume de dados aumenta.

O acúmulo crescente de dados no fragmento de Tempo Real é um dos principais problemas desta proposta. Este fragmento acumula todas as inserções realizadas no Data Warehouse e, para atendê-las satisfatoriamente, não possui nenhum objeto para otimização de leituras. Assim, o seu crescimento irá deteriorar proporcionalmente o tempo de execução destas consultas. Desta forma, deve-se avaliar a queda de desempenho no acesso aos dados do fragmento de tempo real proporcionalmente à quantidade de dados neste fragmento. Ao se realizar este teste, verifica-se a partir de qual volume de dados será necessário redistribuí-los do fragmento de tempo real para os fragmentos dados históricos.

Também deve-se verificar a evolução dos tempos das cargas conforme ocorre o crescimento do fragmento de tempo real, para poder avaliar se as cargas contínuas ocorrerão sempre com desempenho similar ou se o volume de dados já inseridos no fragmento pode ser um fator de perda de desempenho das mesmas.

Assim, a proposta será viável se os impactos causados pelo crescimento do fragmento de tempo real não tornar necessário um intervalo de redistribuição dos dados curto a ponto de afetar o desempenho do Data Warehouse. Já que tal execução envolve a inserção de dados nos fragmentos históricos, afetando a execução das consultas nos mesmos.

A **análise do tempo de redistribuição dos dados** mede o tempo necessário para redistribuir os dados do fragmento de tempo real para os fragmentos históricos. Esta análise permite avaliar a janela necessária de indisponibilidade do Data Warehouse, validando se a mesma seria viável.

Por fim, também são efetuadas algumas comparações com uma das propostas existentes, apresentadas na Seção 3.2.

5.5. Resultados Obtidos

Esta seção apresenta os resultados dos cenários descritos na Seção 5.3. A duração das consultas para os cenários analisados são apresentados em tabelas,

assim como gráficos comparativos entre o tempo de execução das consultas na ausência de cargas e durante as suas realizações.

5.5.1. Tempo Consultas

As seções 5.5.1.1 e 5.5.1.2 apresentam tabelas e gráficos referentes aos tempos de execução de consultas para o cenário tradicional e proposto, respectivamente, enquanto que a seção 5.5.1.3 apresenta uma análise comparativa destes cenários.

5.5.1.1. Arquitetura Tradicional

A Tabela 5 e a Figura 44 apresentam os valores em milissegundos do tempo de execução das consultas descritas no SSB para o cenário tradicional descrito na Seção 4.1. Considerando este cenário, a execução de consultas durante a carga de dados é em média 32,7% mais lenta que a execução da consulta fora do período de cargas.

Tabela 5 – Tempos Cenário Tradicional

	Sem Carga (ms)	Com Carga (ms)
1.1	338.356	509.441
1.2	339.215	499.358
1.3	338.665	494.028
2.1	342.948	511.675
2.2	341.939	508.571
2.3	342.424	499.849
3.1	347.171	521.807
3.2	346.862	514.605
3.3	347.185	505.951
3.4	653.665	987.273
4.1	348.105	525.053
4.2	349.134	520.393
4.3	348.808	520.030

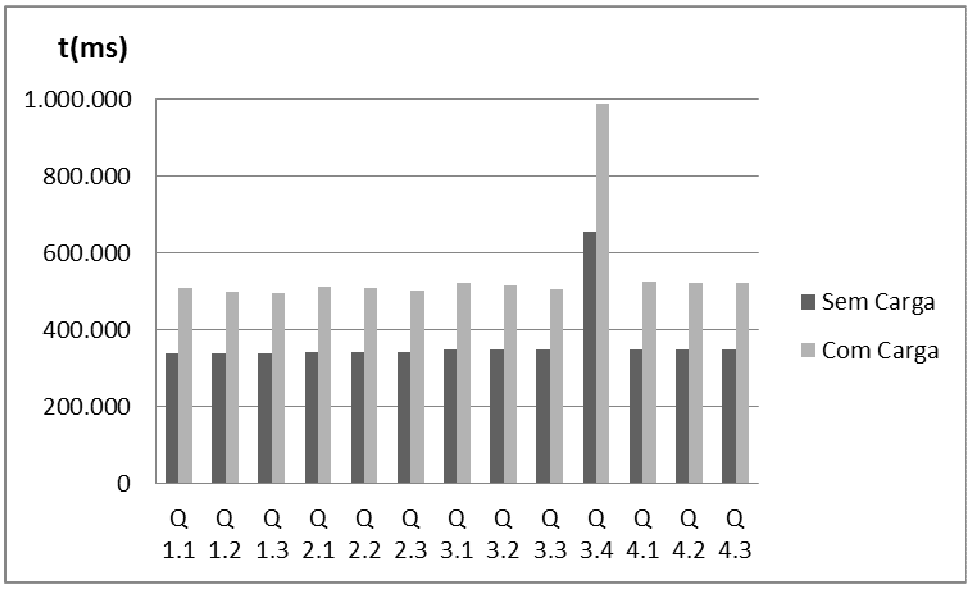


Figura 44 - Tempos Consultas Arquitetura Tradicional

5.5.1.2. Arquitetura Proposta

A Tabela 6 e a Figura 45 apresentam os valores em milissegundos do tempo de execução das consultas descritas no SSB para o cenário proposto por este trabalho, descrito na Seção 4.2. Observe que o tempo é praticamente o mesmo tanto enquanto consultas são realizadas sem cargas simultâneas ou enquanto há cargas simultâneas.

Tabela 6 – Tempos Cenário Proposto

	Sem Carga (ms)	Com Carga (ms)
1.1	369.155	367.229
1.2	368.135	368.041
1.3	368.674	368.657
2.1	370.658	372.378
2.2	371.652	371.691
2.3	370.376	370.592
3.1	375.204	376.616
3.2	376.540	374.771
3.3	377.518	376.315
3.4	816.512	817.142
4.1	378.001	378.183
4.2	379.245	379.285
4.3	379.069	378.051

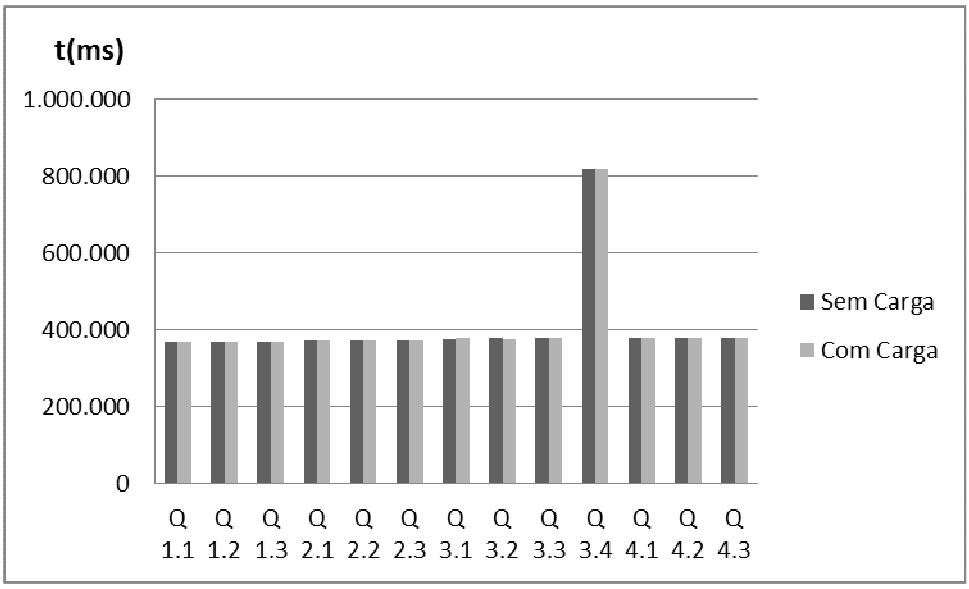


Figura 45 - Tempos Consultas Arquitetura Proposta

5.5.1.3. Comparação entre Arquiteturas

A partir dos valores apresentados nas Seções 5.5.1.1 e 5.5.1.2 ilustrados na Figura 46, observa-se que em um cenário de Data Warehousing tradicional, onde cargas não acontecem em conjunto com a execução de consultas analíticas, consultas realizadas considerando a fragmentação tradicional obtém um desempenho médio superior à fragmentação proposta de aproximadamente 9,8%, se destacando a consulta 3.4 onde o desempenho é superior em 24,9%.

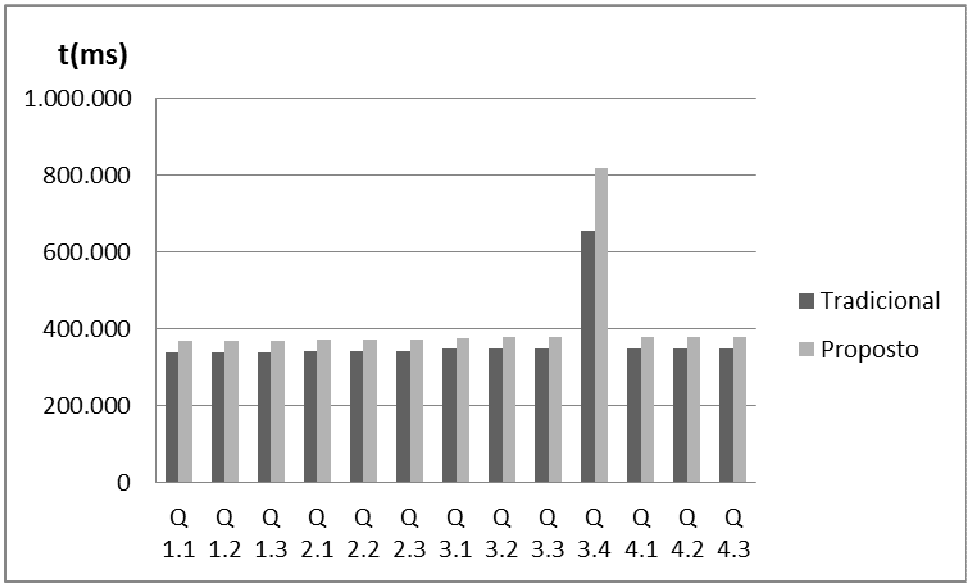


Figura 46 – Comparação Tempos Sem Cargas

Porém, assim que cargas são realizadas em conjunto com as consultas, o tempo de execução das mesmas cresce no cenário tradicional, enquanto que no cenário proposto se mantém praticamente inalterados. Assim as consultas no cenário

proposto ocorrem com um desempenho em média 35,6% superior. A Figura 47 apresenta esta comparação.

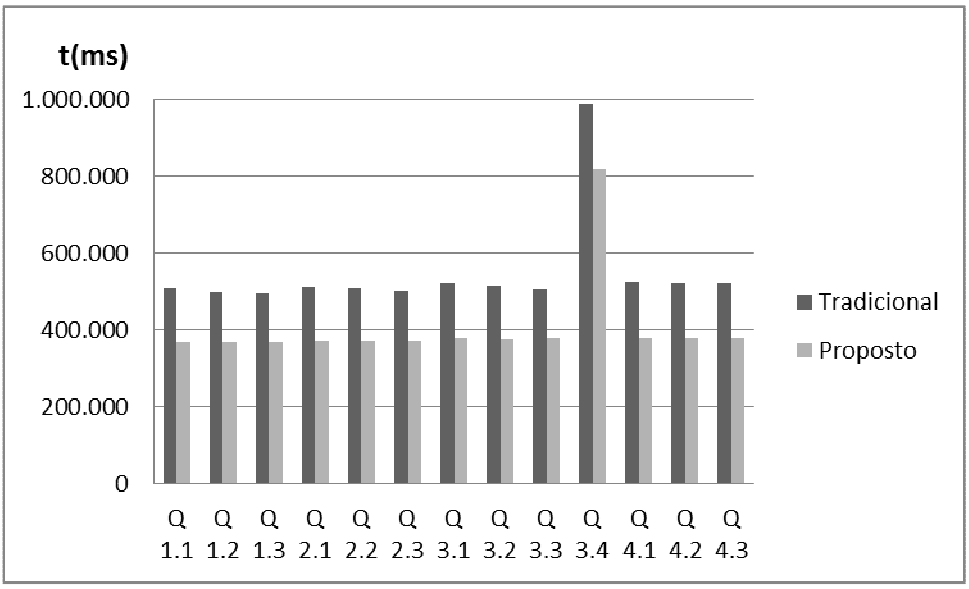


Figura 47 – Comparação Tempos Durante as Cargas

Outro ponto a ser avaliado é o tempo necessário para a inserção completa de um arquivo de carga. Como tais arquivos possuem transações equivalentes a um dia de negócio, o seu tempo de inserção não refletirá a latência da arquitetura, já que esta depende diretamente da estratégia de Carga em Tempo Real adotada, e sim o desempenho do Data Warehouse na execução de cargas contínuas.

As cargas são efetuadas simultaneamente à execução de consultas a fim de simular o comportamento do ambiente de Real Time Data Warehousing.

Neste quesito, o cenário proposto apresentou um desempenho bastante superior nas cargas, conforme a Figura 48 apresenta. Este comportamento se deve ao destino das inserções na arquitetura proposta enfrentar concorrência com consultas menos custosas, devido à menor massa de dados do fragmento de tempo real, além da ausência de estruturas como índices e visões materializadas.

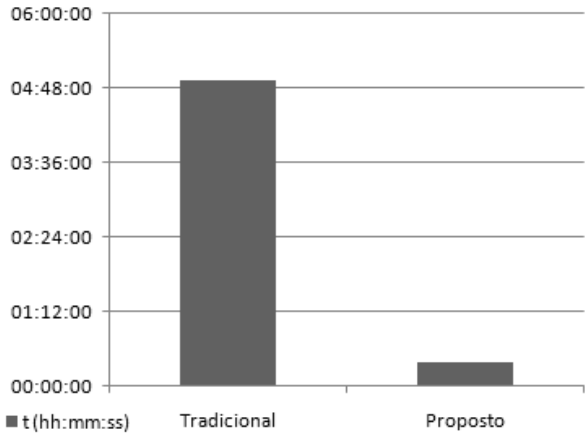


Figura 48 - Comparação Tempos das Cargas

Tais resultados demonstram que a solução proposta atende ao desafio de Real Time Data Warehousing, apresentando um ganho significativo perante a solução tradicional de fragmentação de dados.

5.5.2. Desempenho no Fragmento de Tempo Real

Enquanto os resultados apresentados na Seção 5.5.1 demonstram que a solução proposta permite a realização de cargas contínuas simultâneas à execução de consultas, ainda é necessário comprovar a viabilidade desta solução com o passar do tempo.

Assim, na Tabela 7 e na Figura 49 são apresentados os tempos de execução das consultas Q5.1 à Q5.4 conforme ocorre o crescimento de volume de dados do fragmento de tempo real. Tal crescimento é simbolizado no gráfico pela conclusão de uma tarefa de carga proposta no benchmark. Cada tarefa de carga simboliza uma inserção com o valor de 0,1% dos dados da tabela fatos inicial, no caso seiscentos mil registros, logo as consultas relacionadas à Carga 40 estarão acessando vinte e quatro milhões de tuplas.

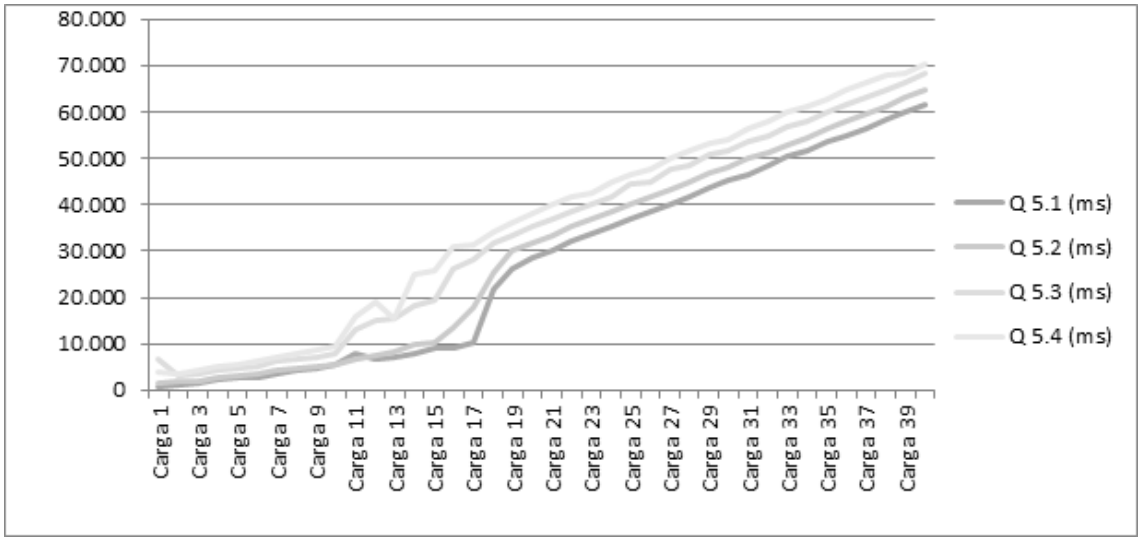


Figura 49 - Impacto nas consultas do fragmento de tempo real

Tabela 7 - Tempos das consultas no fragmento de tempo real

	Q 5.1 (ms)	Q 5.2 (ms)	Q 5.3 (ms)	Q 5.4 (ms)
Carga 1	577	1.541	6.663	3.726
Carga 2	1.056	1.727	2.943	3.475
Carga 3	1.562	2.023	3.574	4.302
Carga 4	2.193	2.731	4.132	4.952
Carga 5	2.641	3.086	4.559	5.652
Carga 6	2.772	3.652	5.099	6.408
Carga 7	3.608	4.276	6.093	7.121
Carga 8	4.393	4.806	6.520	7.915
Carga 9	4.792	5.233	7.164	8.663
Carga 10	5.326	5.472	7.951	9.590
Carga 11	8.017	6.567	13.211	16.022
Carga 12	6.782	7.448	14.906	18.931
Carga 13	7.278	8.360	15.517	15.516
Carga 14	7.803	9.772	18.223	24.843
Carga 15	9.012	10.315	19.463	25.747
Carga 16	8.948	13.493	26.318	30.850
Carga 17	10.354	18.004	28.343	31.361
Carga 18	21.714	25.483	31.655	34.036
Carga 19	26.185	29.972	33.465	35.946
Carga 20	28.681	31.710	35.487	38.180
Carga 21	30.174	33.239	36.723	40.049
Carga 22	32.163	35.271	38.575	41.548
Carga 23	33.714	36.769	40.059	42.530
Carga 24	35.363	38.474	41.865	44.803
Carga 25	36.732	40.156	44.279	46.576
Carga 26	38.570	41.686	45.049	47.615
Carga 27	40.188	43.119	47.502	49.904
Carga 28	41.659	44.735	48.293	51.769
Carga 29	43.495	46.829	50.988	53.022
Carga 30	45.136	48.231	51.477	54.177
Carga 31	46.410	49.874	53.706	56.562
Carga 32	48.608	51.371	54.877	58.010
Carga 33	50.290	52.917	56.887	59.882
Carga 34	51.639	54.568	58.179	61.044
Carga 35	53.489	56.354	59.829	62.942
Carga 36	55.000	58.081	61.626	64.772
Carga 37	56.566	59.608	62.982	66.256
Carga 38	58.386	61.351	64.610	67.757
Carga 39	60.096	63.060	66.149	68.415
Carga 40	61.663	64.937	68.143	70.251

Na Figura 49, observa-se que após a execução de quarenta cargas, ou seja, quando o fragmento de tempo real possuir aproximadamente 4% do tamanho da tabela fatos original, o tempo máximo da consulta foi de 70 segundos. Tal valor ainda

é bastante inferior ao tempo médio da execução das consultas nos dados históricos, que era de 370 segundos.

Vale salientar que a tabela fatos gerada inicialmente abrange um período de sete anos, assim, tal quantidade de cargas executadas estará simulando inserções realizadas durante diversos dias sem a redistribuição do fragmento. Logo, neste caso simulado, redistribuições de caráter diário devem atender satisfatoriamente às necessidades de limpeza do fragmento de tempo real.

Outro ponto a ser avaliado na proposta para confirmar sua viabilidade é se ocorrerá alguma queda no desempenho das cargas conforme o volume de dados do fragmento de tempo real cresce. Para atender a tal questionamento, na Tabela 8 são apresentados os tempos das quarenta cargas executadas durante o experimento anterior.

Tabela 8 - Tempos de execução das cargas contínuas em seqüência

	T Carga (s)		T Carga (s)
Carga 1	1.361	Carga 21	1.290
Carga 2	1.431	Carga 22	1.389
Carga 3	1.304	Carga 23	1.316
Carga 4	1.399	Carga 24	1.382
Carga 5	1.466	Carga 25	1.439
Carga 6	1.420	Carga 26	1.326
Carga 7	1.402	Carga 27	1.406
Carga 8	1.345	Carga 28	1.358
Carga 9	1.346	Carga 29	1.420
Carga 10	1.364	Carga 30	1.313
Carga 11	1.384	Carga 31	1.287
Carga 12	1.376	Carga 32	1.332
Carga 13	1.353	Carga 33	1.367
Carga 14	1.372	Carga 34	1.298
Carga 15	1.406	Carga 35	1.304
Carga 16	1.394	Carga 36	1.328
Carga 17	1.403	Carga 37	1.301
Carga 18	1.373	Carga 38	1.283
Carga 19	1.327	Carga 39	1.454
Carga 20	1.412	Carga 40	1.485

A Figura 50 demonstra que o crescimento do volume dos dados no fragmento de tempo real não afeta o desempenho das cargas, já que o tempo oscila entre 22 e 24 minutos, sem aparente relação com o volume de dados do fragmento.

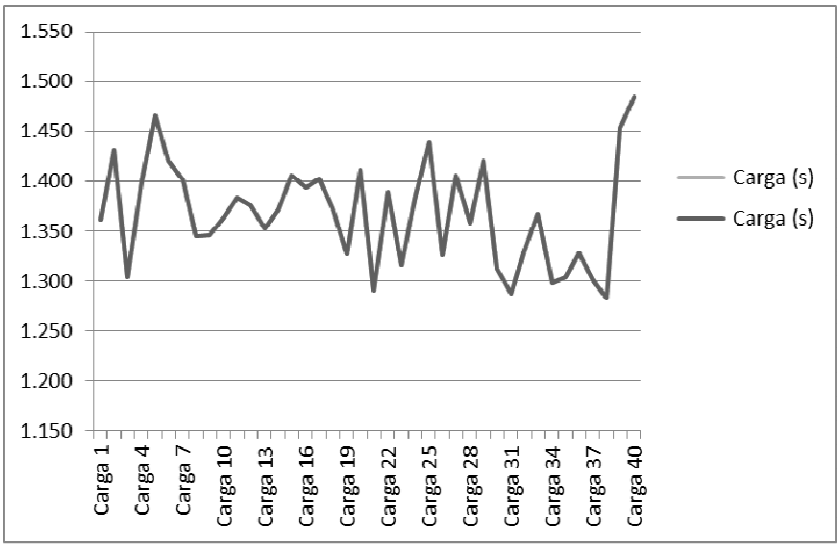


Figura 50 - Comparação entre os tempos de execução das cargas contínuas

Assim, os problemas causados pela inserção contínua dos dados exclusivamente no fragmento de tempo real podem ser tratados com a redistribuição dos dados em intervalos convenientes, viabilizando a execução da proposta.

5.5.3. Redistribuição dos dados inseridos

Uma tarefa essencial para a viabilidade desta proposta é a redistribuição dos dados inseridos no fragmento de tempo real para os fragmentos históricos.

Os testes da Seção 5.5.2 forneceram a conclusão de que uma periodicidade diária de redistribuição seria aceitável para o SSB. Assim, esta tarefa pode ocorrer em janelas de indisponibilidade sem concorrer com a execução de consultas de usuários.

Considerando isso, foram executados mapas ETLs para redistribuir os dados para os fragmentos históricos sem a concorrência de consultas. Os resultados são apresentados na Tabela 9. Cada entrada desta tabela simboliza o tempo necessário para executar o procedimento de redistribuição em uma massa de dados igual a um arquivo de carga proposto no *benchmark*, ou seja, 0,1% das linhas da tabela fatos histórica.

Tabela 9 – Tempos para redistribuição dos dados do fragmento de tempo real

	Tempo
Redistribuição após Carga 1	00:36:29
Redistribuição após Carga 2	00:25:58
Redistribuição após Carga 3	00:24:54
Redistribuição após Carga 4	00:26:39
Redistribuição após Carga 5	00:25:17

Nos resultados, a redistribuição de uma massa de dados igual a 0,5% das linhas da tabela fatos histórica demorou 02h 19m 16s. Seguindo as especificações do benchmark, a quantidade de transações desta massa de dados supera a de um dia de negócio comum. Logo, a execução da tarefa de redistribuição dos dados ocorrerá em um intervalo médio de 2 horas, valor bastante aceitável para a indisponibilidade de um DW cuja arquitetura tradicionalmente prevê indisponibilidades para inserção dos dados a partir das fontes operacionais.

5.5.4. Comparação com outras propostas

Diversas outras propostas de Real Time DW foram discutidas na literatura e apresentadas na Seção 3.2. Entretanto, tais propostas não apresentam cenários que envolvam a distribuição de dados, principal inovação do cenário descrito nesta dissertação.

Logo, realizar a comparação entre tais propostas considerando apenas tempos de execução obtidos por um benchmark iria ser diretamente afetado pela capacidade computacional que a distribuição fornece à proposta aqui apresentada.

Assim, o foco da comparação realizada foi apresentar as deficiências existentes em algumas propostas atuais e demonstrar como elas afetam o usuário e o desempenho da proposta, além de como este trabalho consegue superá-la.

Para tanto, escolhemos uma das propostas existentes, a de SANTOS e BERNARDINO (2009). Sua principal deficiência é a necessidade de que as ferramentas de BI tenham conhecimento da localização dos dados históricos e atuais, os quais são armazenados em objetos diferentes. Tais problemas foram solucionados nesta proposta com o uso de um SGBDD, o qual traduz as requisições das ferramentas para os destinos necessários.

Na proposta de SANTOS e BERNARDINO (2009), os dados são separados entre dois tipos de objetos, um conjunto de tabelas para dados históricos e um conjunto de tabelas para dados inseridos em tempo real. Para executar consultas nesta proposta é necessário acessar os dois conjuntos de objetos e utilizar o operador SQL UNION para fornecer a resposta final.

Um dos principais problemas causados por esta abordagem é o aumento significativo da complexidade das consultas. Com a necessidade de digitar mais código SQL para acesso aos dados, que passam a existir em uma quantidade maior de objetos, o trabalho de escrever instruções SQL para estas propostas piora o já complexo cenário de consultas em Data Warehouses.

Para avaliar tal afirmação, foram utilizadas as consultas do benchmark SSB, descrito na Seção 5.1. Como o mesmo é composto por quatro categorias de consultas, cada qual possuindo consultas similares que simbolizam um tipo específico de operação, foi escolhida uma consulta de cada um destes grupos para serem adaptadas a fim de permitir sua execução na proposta de SANTOS e BERNARDINO (2009).

A Figura 51, a Figura 53, a Figura 55 e a Figura 57 apresentam as consultas escolhidas em sua forma original. Já a Figura 52, a Figura 54, a Figura 56 e a Figura 58 apresentam as respectivas adaptações para execução na proposta de SANTOS e BERNARDINO (2009). Nelas é claramente perceptível o aumento de complexidade das consultas, cuja quantidade de instruções aproximadamente dobra.

```
select sum(lo_extendedprice*lo_discount) as revenue
from lineorder, date
where lo_orderdate = d_datekey
    and d_year = 1993
    and lo_discount between 1 and 3
    and lo_quantity < 25
```

Figura 51 – Consulta Original Q1.1 do SSB

```
select sum(lo_extendedprice*lo_discount) as revenue
from  ( select lo_extendedprice, lo_discount
 from lineorder, date
 where lo_orderdate = d_datekey
      and d_year = 1993
      and lo_discount between 1 and 3
      and lo_quantity < 25
 UNION ALL
 select lo_extendedprice, lo_discount
 from lineorderTmp, dateTmp
 where lo_orderdate = d_datekey
      and d_year = 1993
      and lo_discount between 1 and 3
      and lo_quantity < 25 ) as result
```

Figura 52 – Consulta Q1.1 do SSB adaptada para a proposta de SANTOS e BERNARDINO (2009)

```
select sum(lo_revenue), d_year, p_brand1
from lineorder, date, part, supplier
where lo_orderdate = d_datekey
    and lo_partkey = p_partkey
    and lo_suppkey = s_suppkey
    and p_category = 'MFGR#12'
    and s_region = 'AMERICA'
group by d_year, p_brand1
order by d_year, p_brand1
```

Figura 53 – Consulta Original Q2.1 do SSB

```
select sum(lo_revenue), d_year, p_brand1
from  ( select lo_revenue, d_year, p_brand1
 from lineorder, date, part, supplier
 where lo_orderdate = d_datekey
      and lo_partkey = p_partkey
      and lo_suppkey = s_suppkey
      and p_category = 'MFGR#12'
      and s_region = 'AMERICA'
 UNION ALL
 select lo_revenue, d_year, p_brand1
 from lineorderTmp, dateTmp, partTmp, supplierTmp
 where lo_orderdate = d_datekey
      and lo_partkey = p_partkey
      and lo_suppkey = s_suppkey
      and p_category = 'MFGR#12'
      and s_region = 'AMERICA') as result
group by d_year, p_brand1
order by d_year, p_brand1
```

Figura 54 – Consulta Q2.1 do SSB adaptada para a proposta de SANTOS e BERNARDINO (2009)

```
select c_nation, s_nation, d_year, sum(lo_revenue) as
revenue
from customer, lineorder, supplier, date
where lo_custkey = c_custkey
    and lo_suppkey = s_suppkey
    and lo_orderdate = d_datekey
    and c_region = 'ASIA'
    and s_region = 'ASIA'
    and d_year >= 1992 and d_year <= 1997
group by c_nation, s_nation, d_year
order by d_year asc, sum(lo_revenue) desc
```

Figura 55 – Consulta Original Q3.1 do SSB

```
select c_nation,s_nation,d_year,sum(lo_revenue) as revenue
from  ( select c_nation, s_nation, d_year, lo_revenue
 from customer, lineorder, supplier, date
 where lo_custkey = c_custkey
      and lo_suppkey = s_suppkey
      and lo_orderdate = d_datekey
      and c_region = 'ASIA'
      and s_region = 'ASIA'
      and d_year >= 1992 and d_year <= 1997
 UNION ALL
 select c_nation, s_nation, d_year, lo_revenue
 from customerTmp, lineorderTmp, supplierTmp, dateTmp
 where lo_custkey = c_custkey
      and lo_suppkey = s_suppkey
      and lo_orderdate = d_datekey
      and c_region = 'ASIA'
      and s_region = 'ASIA'
      and d_year >= 1992 and d_year <= 1997) as result
group by c_nation, s_nation, d_year
order by d_year asc, sum(lo_revenue) desc
```

Figura 56 – Consulta Q3.1 do SSB adaptada para a proposta de SANTOS e BERNARDINO (2009)

```
select d_year, c_nation, sum(lo_revenue - lo_supplycost)
as profit
from date, customer, supplier, part, lineorder
where lo_custkey = c_custkey
    and lo_suppkey = s_suppkey
    and lo_partkey = p_partkey
    and lo_orderdate = d_datekey
    and c_region = 'AMERICA' and s_region = 'AMERICA'
    and (p_mfgr = 'MFGR#1' or p_mfgr = 'MFGR#2')
group by d_year, c_nation
order by d_year, c_nation;
```

Figura 57 – Consulta Original Q4.1 do SSB

```
select d_year, c_nation, sum(lo_revenue - lo_supplycost)
as profit
from  ( select d_year, c_nation, lo_revenue, lo_supplycost
 from date, customer, supplier, part, lineorder
 where lo_custkey = c_custkey
       and lo_suppkey = s_suppkey
       and lo_partkey = p_partkey
       and lo_orderdate = d_datekey
       and c_region = 'AMERICA' and s_region = 'AMERICA'
       and (p_mfgr = 'MFGR#1' or p_mfgr = 'MFGR#2')
 UNION ALL
 select d_year, c_nation, lo_revenue, lo_supplycost
 from   dateTmp,   customerTmp,   supplierTmp,   partTmp,
lineorderTmp
 where lo_custkey = c_custkey
       and lo_suppkey = s_suppkey
       and lo_partkey = p_partkey
       and lo_orderdate = d_datekey
       and c_region = 'AMERICA' and s_region = 'AMERICA'
       and (p_mfgr = 'MFGR#1' or p_mfgr = 'MFGR#2')) as
result
group by d_year, c_nation
order by d_year, c_nation
```

Figura 58 – Consulta Q4.1 do SSB adaptada para a proposta de SANTOS e BERNARDINO (2009)

Outro impacto que este aumento de complexidade ocasiona é refletido no próprio SGBD, já que os planos de execução elaborados pelos planejadores dos bancos passam a ter custos mais elevados.

Utilizando o comando EXPLAIN do PostgreSQL, foi verificado o custo associado na execução das consultas do SSB escolhidas acima. Este custo indica o quanto o planejador acredita que determinada instrução levará para executar e é medido em acessos necessários a páginas de disco. (POSTGRESQL, 2010)

A Figura 59, a Figura 61, a Figura 63 e a Figura 65 apresentam o plano de execução para as consultas originais do SSB, enquanto que a Figura 60, a Figura 62, a Figura 64 e a Figura 66 apresentam o plano de execução para as consultas

adaptadas para a proposta de SANTOS e BERNARDINO (2009). Nelas é perceptível o aumento de custo da consulta e de etapas do plano de execução quando a mesma necessita ser adaptada para refletir a localidade dos dados.

```
Aggregate  (cost=4856615.10..4856615.11 rows=1 width=14)
  ->  Hash Join  (cost=86.51..4850599.51 rows=2406233 width=14)
       Hash Cond: (lineorder.lo_orderdate = date.d_datekey)
       -> Seq Scan on lineorder (cost=0.00..4766970.56 rows=15861362 width=23)
            Filter: ((lo_discount >= 1::numeric) AND (lo_discount <= 3::numeric)
            AND (lo_quantity < 25::numeric))
       ->  Hash  (cost=81.95..81.95 rows=365 width=9)
            ->  Seq Scan on date  (cost=0.00..81.95 rows=365 width=9)
                 Filter: (d_year = 1993::numeric)
```

Figura 59 – Plano de execução da Consulta Q1.1 do SSB

```
Aggregate  (cost=5139137.31..5139137.33 rows=1 width=64)
  ->  Append  (cost=86.51..5131120.21 rows=3206840 width=64)
       ->  Hash Join  (cost=86.51..4850599.51 rows=2406233 width=14)
            Hash Cond: (lineorder.lo_orderdate = date.d_datekey)
            ->SeqScan on lineorder(cost=0.00..4766970.56 rows=15861362 width=23)
                 Filter: ((lo_discount >= 1::numeric) AND (lo_discount <= 3::numeric)
                  AND (lo_quantity < 25::numeric))
            ->  Hash  (cost=81.95..81.95 rows=365 width=9)
                 ->  Seq Scan on date  (cost=0.00..81.95 rows=365 width=9)
                      Filter: (d_year = 1993::numeric)
       ->  Nested Loop  (cost=0.00..248452.30 rows=800607 width=14)
            Join Filter: (lineordertmp.lo_orderdate = datetmp.d_datekey)
            ->  Seq Scan on datetmp  (cost=0.00..11.88 rows=1 width=32)
                 Filter: (d_year = 1993::numeric)
            ->SeqScan on lineordertmp(cost=0.00..238432.84 rows=800607 width=23)
                 Filter: ((lineordertmp.lo_discount >= 1::numeric)
                  AND (lineordertmp.lo_discount <= 3::numeric)
                  AND (lineordertmp.lo_quantity < 25::numeric))
```

Figura 60 – Plano de execução da Consulta Q1.1 adaptada para SANTOS e BERNARDINO (2009)

```
Sort  (cost=5472931.02..5472931.69 rows=269 width=25)
  Sort Key: date.d_year, part.p_brand1
  ->  HashAggregate  (cost=5472916.80..5472920.16 rows=269 width=25)
      ->  Hash Join  (cost=46387.51..5462575.40 rows=1378853 width=25)
          Hash Cond: (lineorder.lo_orderdate = date.d_datekey)
          ->  Hash Join  (cost=46280.00..5441785.10 rows=1378853 width=27)
              Hash Cond: (lineorder.lo_suppkey = supplier.s_suppkey)
              ->  Hash Join  (cost=39856.00..5322742.13 rows=6588695 width=35)
                  Hash Cond: (lineorder.lo_partkey = part.p_partkey)
                  ->  Seq Scan on lineorder
                              (cost=0.00..3866913.32 rows=120007632 width=33)
                  ->  Hash  (cost=39184.00..39184.00 rows=53760 width=18)
                      ->SeqScan on part (cost=0.00..39184.00 rows=53760 width=18)
                          Filter: (p_category = 'MFGR#12'::bpchar)
              ->  Hash  (cost=5922.00..5922.00 rows=40160 width=8)
                  ->Seq Scan on supplier  (cost=0.00..5922.00 rows=40160 width=8)
                      Filter: (s_region = 'AMERICA'::bpchar)
          ->  Hash  (cost=75.56..75.56 rows=2556 width=16)
              ->  Seq Scan on date  (cost=0.00..75.56 rows=2556 width=16)
```

Figura 61 – Plano de execução da Consulta Q2.1 do SSB

```
Sort  (cost=5706225.04..5706325.04 rows=40000 width=104)
  Sort Key: date.d_year, part.p_brand1
  ->  HashAggregate  (cost=5702667.50..5703167.50 rows=40000 width=104)
      ->  Append  (cost=46387.51..5692326.09 rows=1378854 width=104)
          ->  Hash Join  (cost=46387.51..5462575.40 rows=1378853 width=25)
              Hash Cond: (lineorder.lo_orderdate = date.d_datekey)
              ->  Hash Join  (cost=46280.00..5441785.10 rows=1378853 width=27)
                  Hash Cond: (lineorder.lo_suppkey = supplier.s_suppkey)
                  ->Hash Join(cost=39856.00..5322742.13 rows=6588695 width=35)
                      Hash Cond: (lineorder.lo_partkey = part.p_partkey)
                      ->  Seq Scan on lineorder
                              (cost=0.00..3866913.32 rows=120007632 width=33)
                      ->  Hash  (cost=39184.00..39184.00 rows=53760 width=18)
                          ->  Seq Scan on part
                                  (cost=0.00..39184.00 rows=53760 width=18)
                              Filter: (p_category = 'MFGR#12'::bpchar)
                  ->  Hash  (cost=5922.00..5922.00 rows=40160 width=8)
                      ->  Seq Scan on supplier
                              (cost=0.00..5922.00 rows=40160 width=8)
                          Filter: (s_region = 'AMERICA'::bpchar)
              ->  Hash  (cost=75.56..75.56 rows=2556 width=16)
                  ->  Seq Scan on date  (cost=0.00..75.56 rows=2556 width=16)
          ->  Nested Loop  (cost=12.14..215962.15 rows=1 width=80)
              Join Filter: (lineordertmp.lo_orderdate = datetmp.d_datekey)
              ->  Nested Loop  (cost=12.14..215948.78 rows=1 width=57)
                  Join Filter: (lineordertmp.lo_partkey = parttmp.p_partkey)
                  ->  Seq Scan on parttmp  (cost=0.00..12.50 rows=1 width=72)
                      Filter: (p_category = 'MFGR#12'::bpchar)
                  ->  Hash Join  (cost=12.14..215935.89 rows=31 width=25)
                      HashCond:(lineordertmp.lo_suppkey = suppliertmp.s_suppkey)
                      ->  Seq Scan on lineordertmp
                              (cost=0.00..193414.05 rows=6002505 width=33)
                      ->  Hash  (cost=12.12..12.12 rows=1 width=32)
                          ->  Seq Scan on suppliertmp
                              (cost=0.00..12.12 rows=1 width=32)
                              Filter: (s_region = 'AMERICA'::bpchar)
              ->  Seq Scan on datetmp  (cost=0.00..11.50 rows=150 width=64)
```

Figura 62 – Plano de execução da Consulta Q2.1 adaptada para SANTOS e BERNARDINO (2009)

```
Sort  (cost=6732525.24..6732525.62 rows=149 width=47)
  Sort Key: date.d_year, (sum(lineorder.lo_revenue))
  ->  HashAggregate  (cost=6732518.00..6732519.87 rows=149 width=47)
      ->  Hash Join  (cost=106839.15..6633084.49 rows=9943351 width=47)
          Hash Cond: (lineorder.lo_orderdate = date.d_datekey)
          ->  Hash Join  (cost=106723.41..6478964.75 rows=10914098 width=49)
              Hash Cond: (lineorder.lo_custkey = customer.c_custkey)
              ->  Hash Join  (cost=6423.66..5924429.38 rows=25097792 width=41)
                  Hash Cond: (lineorder.lo_suppkey = supplier.s_suppkey)
                  ->  Seq Scan on lineorder
                              (cost=0.00..3866913.32 rows=120007632 width=33)
                  ->  Hash  (cost=5922.00..5922.00 rows=40133 width=24)
                      ->  Seq Scan on supplier
                              (cost=0.00..5922.00 rows=40133 width=24)
                          Filter: (s_region = 'ASIA'::bpchar)
              ->  Hash  (cost=92881.00..92881.00 rows=593500 width=24)
                  ->  Seq Scan on customer
                              (cost=0.00..92881.00 rows=593500 width=24)
                      Filter: (c_region = 'ASIA'::bpchar)
          ->  Hash  (cost=88.34..88.34 rows=2192 width=16)
              ->  Seq Scan on date  (cost=0.00..88.34 rows=2192 width=16)
                  Filter: ((d_year >= 1992::numeric) AND (d_year <= 1997::numeric))
```

Figura 63 – Plano de execução da Consulta Q3.1 do SSB

```
Sort  (cost=9349322.61..9351808.45 rows=994336 width=192)
  Sort Key: date.d_year, (sum(lineorder.lo_revenue))
  ->  GroupAggregate  (cost=9021785.87..9158506.97 rows=994336 width=192)
      ->  Sort  (cost=9021785.87..9046644.25 rows=9943352 width=192)
          Sort Key: customer.c_nation, supplier.s_nation, date.d_year
          ->  Result  (cost=106839.15..6948478.33 rows=9943352 width=192)
              ->  Append  (cost=106839.15..6948478.33 rows=9943352 width=192)
                 ->HashJoin (cost=106839.15..6633084.49 rows=9943351 width=47)
                       Hash Cond: (lineorder.lo_orderdate = date.d_datekey)
                       -> Hash Join
                             (cost=106723.41..6478964.75 rows=10914098 width=49)
                          Hash Cond: (lineorder.lo_custkey = customer.c_custkey)
                          -> Hash Join
                                (cost=6423.66..5924429.38 rows=25097792 width=41)
                             HashCond:(lineorder.lo_suppkey = supplier.s_suppkey)
                             ->  Seq Scan on lineorder
                              (cost=0.00..3866913.32 rows=120007632 width=33)
                             ->Hash  (cost=5922.00..5922.00 rows=40133 width=24)
                                ->  Seq Scan on supplier
                                          (cost=0.00..5922.00 rows=40133 width=24)
                                   Filter: (s_region = 'ASIA'::bpchar)
                          ->Hash  (cost=92881.00..92881.00 rows=593500 width=24)
                             ->  Seq Scan on customer
                                      (cost=0.00..92881.00 rows=593500 width=24)
                                Filter: (c_region = 'ASIA'::bpchar)
                       ->  Hash  (cost=88.34..88.34 rows=2192 width=16)
                          ->Seq Scan on date(cost=0.00..88.34 rows=2192 width=16)
                             Filter: ((d_year >= 1992::numeric)
                                   AND (d_year <= 1997::numeric))
                    ->  Nested Loop  (cost=12.01..215960.32 rows=1 width=168)
                       Join Filter: (lineordertmp.lo_orderdate = datetmp.d_datekey)
                       ->  Nested Loop  (cost=12.01..215948.05 rows=1 width=145)
                          JoinFilter:(lineordertmp.lo_suppkey = suppliertmp.s_suppkey)
                          ->  Seq Scan on suppliertmp
                                   (cost=0.00..12.12 rows=1 width=96)
                             Filter: (s_region = 'ASIA'::bpchar)
                          ->  Hash Join  (cost=12.01..215935.67 rows=21 width=89)
                             Hash Cond:
                              (lineordertmp.lo_custkey = customertmp.c_custkey)
```

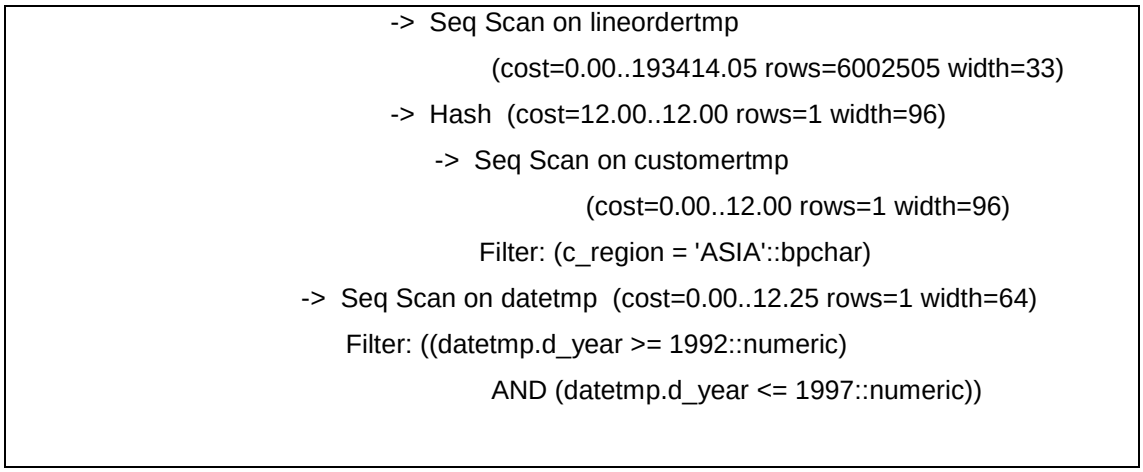


Figura 64 – Plano de execução da Consulta Q3.1 adaptada para SANTOS e BERNARDINO (2009)

```
Sort  (cost=6853294.70..6853294.79 rows=36 width=39)
  Sort Key: date.d_year, customer.c_nation
  ->  HashAggregate  (cost=6853293.23..6853293.77 rows=36 width=39)
      ->  Hash Join  (cost=155882.57..6810813.72 rows=5663935 width=39)
          Hash Cond: (lineorder.lo_orderdate = date.d_datekey)
          ->  Hash Join  (cost=155775.06..6725747.19 rows=5663935 width=41)
              Hash Cond: (lineorder.lo_partkey = part.p_partkey)
              ->Hash Join  (cost=106847.50..6481385.06 rows=11103618 width=49)
                  Hash Cond: (lineorder.lo_custkey = customer.c_custkey)
                  ->Hash Join(cost=6424.00..5924598.57 rows=25114677 width=41)
                      Hash Cond: (lineorder.lo_suppkey = supplier.s_suppkey)
                      ->  Seq Scan on lineorder
                              (cost=0.00..3866913.32 rows=120007632 width=49)
                      ->  Hash  (cost=5922.00..5922.00 rows=40160 width=8)
                          ->  Seq Scan on supplier
                                  (cost=0.00..5922.00 rows=40160 width=8)
                              Filter: (s_region = 'AMERICA'::bpchar)
                  ->  Hash  (cost=92881.00..92881.00 rows=603400 width=24)
                      ->  Seq Scan on customer
                              (cost=0.00..92881.00 rows=603400 width=24)
                          Filter: (c_region = 'AMERICA'::bpchar)
              ->  Hash  (cost=42684.00..42684.00 rows=499485 width=8)
                  ->  Seq Scan on part  (cost=0.00..42684.00 rows=499485 width=8)
                      Filter: ((p_mfgr = 'MFGR#1'::bpchar)
                              OR (p_mfgr = 'MFGR#2'::bpchar))
          ->  Hash  (cost=75.56..75.56 rows=2556 width=16)
              ->  Seq Scan on date  (cost=0.00..75.56 rows=2556 width=16)
```

Figura 65 – Plano de execução da Consulta Q4.1 do SSB

```
Sort  (cost=7129564.60..7129664.60 rows=40000 width=160)
  Sort Key: date.d_year, customer.c_nation
  ->  HashAggregate  (cost=7125907.05..7126507.05 rows=40000 width=160)
      ->  Append  (cost=155882.57..7083427.53 rows=5663936 width=160)
          ->  Hash Join  (cost=155882.57..6810813.72 rows=5663935 width=39)
              Hash Cond: (lineorder.lo_orderdate = date.d_datekey)
              ->  Hash Join  (cost=155775.06..6725747.19 rows=5663935 width=41)
                  Hash Cond: (lineorder.lo_partkey = part.p_partkey)
                ->Hash Join(cost=106847.50..6481385.06 rows=11103618 width=49)
                      Hash Cond: (lineorder.lo_custkey = customer.c_custkey)
                  ->Hash Join(cost=6424.00..5924598.57 rows=25114677 width=41)
                        Hash Cond: (lineorder.lo_suppkey = supplier.s_suppkey)
                        ->  Seq Scan on lineorder
                            (cost=0.00..3866913.32 rows=120007632 width=49)
                        ->  Hash  (cost=5922.00..5922.00 rows=40160 width=8)
                            ->  Seq Scan on supplier
                                  (cost=0.00..5922.00 rows=40160 width=8)
                                Filter: (s_region = 'AMERICA'::bpchar)
                    ->  Hash  (cost=92881.00..92881.00 rows=603400 width=24)
                        ->  Seq Scan on customer
                                  (cost=0.00..92881.00 rows=603400 width=24)
                            Filter: (c_region = 'AMERICA'::bpchar)
                  ->  Hash  (cost=42684.00..42684.00 rows=499485 width=8)
                    ->  Seq Scan on part (cost=0.00..42684.00 rows=499485 width=8)
                        Filter: ((p_mfgr = 'MFGR#1'::bpchar)
                              OR (p_mfgr = 'MFGR#2'::bpchar))
              ->  Hash  (cost=75.56..75.56 rows=2556 width=16)
                  ->  Seq Scan on date  (cost=0.00..75.56 rows=2556 width=16)
          ->  Nested Loop  (cost=12.01..215974.45 rows=1 width=112)
              Join Filter: (lineordertmp.lo_partkey = parttmp.p_partkey)
              ->  Nested Loop  (cost=12.01..215961.43 rows=1 width=120)
                  Join Filter: (lineordertmp.lo_orderdate = datetmp.d_datekey)
                  ->  Nested Loop  (cost=12.01..215948.05 rows=1 width=97)
                      Join Filter: (lineordertmp.lo_suppkey = suppliertmp.s_suppkey)
                    ->  Seq Scan on suppliertmp  (cost=0.00..12.12 rows=1 width=32)
                        Filter: (s_region = 'AMERICA'::bpchar)
                      ->  Hash Join  (cost=12.01..215935.67 rows=21 width=105)
                    Hash Cond:(lineordertmp.lo_custkey = customertmp.c_custkey)
                        ->  Seq Scan on lineordertmp
```

```
                                (cost=0.00..193414.05 rows=6002505 width=49)
                        ->  Hash  (cost=12.00..12.00 rows=1 width=96)
                            ->  Seq Scan on customertmp
                                    (cost=0.00..12.00 rows=1 width=96)
                                Filter: (c_region = 'AMERICA'::bpchar)
                    ->  Seq Scan on datetmp  (cost=0.00..11.50 rows=150 width=64)
                ->  Seq Scan on parttmp  (cost=0.00..13.00 rows=2 width=32)
                    Filter: ((parttmp.p_mfgr = 'MFGR#1'::bpchar)
                            OR (parttmp.p_mfgr = 'MFGR#2'::bpchar))
```

Figura 66 – Plano de execução da Consulta Q4.1 adaptada para SANTOS e BERNARDINO (2009)

A Figura 67 apresenta graficamente o aumento do custo de execução de cada uma das consultas escolhidas para avaliar o uso da proposta de SANTOS e BERNARDINO (2009). Nela, observa-se um aumento médio de 5% do custo de execução em três consultas (Q1.1, Q2.1 e Q4.1), sendo que a consulta Q3.1 teve um aumento significativo de 39%.

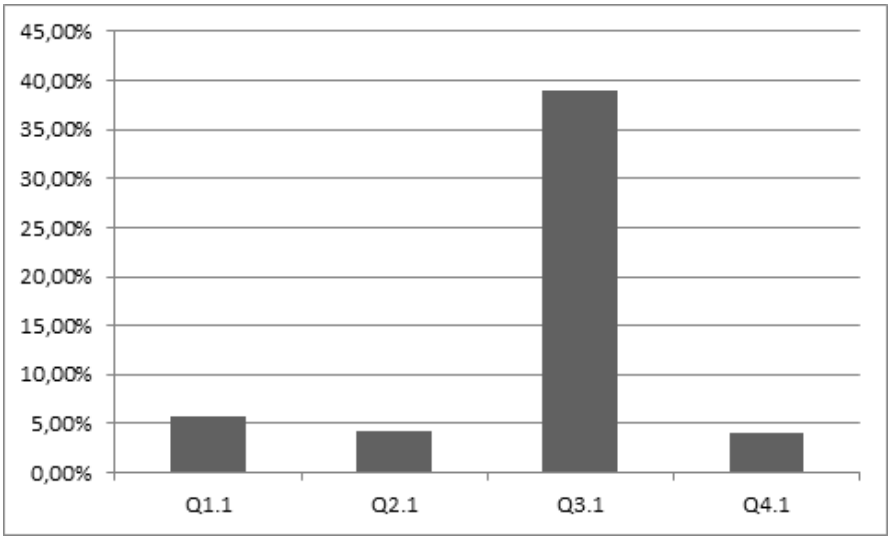


Figura 67 – Aumento porcentual do custo de execução das consultas escolhidas

Vale ressaltar que estas variações no custo de execução podem estar relacionados à utilização de um Sistema Gerenciador de Bancos de Dados distribuídos (pgpool-II) e a forma como ele realiza a união dos fragmentos. Entretanto, este incremento de 5% aliado ao aumento de complexidade das consultas SQL são fatores que demonstram a importância de um mecanismo para visão única dos dados, ocultando a sua localidade para quem os acesse.

6. Conclusões

A popularização e o uso intensivo de ferramentas de BI demandam o surgimento de novas funcionalidades a fim de aprimorar o processo de tomada de decisão. Adicionalmente, novos modelos de negócio surgem constantemente, cada vez mais dinâmicos e globalizados, e com características bem distintas dos modelos tradicionais, demandando mudanças na arquitetura de BI tradicional. A evolução desta arquitetura de BI gerou o chamado BI 2.0. Desafios como uma maior permeabilidade das ferramentas de BI, melhor gerenciamento do desempenho do negócio e análise de dados operacionais em tempo real são algumas das promessas desta nova arquitetura.

O uso de dados operacionais em tempo real, objetivo deste trabalho, prova ser uma área de grande complexidade e que está em constante evolução, já que envolve bancos de dados gigantescos, transformações e integrações complexas de dados e consultas *ad hoc*. Este trabalho buscou ser uma alternativa viável para esta área, objetivando a execução contínua de cargas no DW sem afetar o desempenho de consultas analisando tanto dados históricos quanto recentes.

Neste capítulo de conclusão são apresentadas as considerações finais sobre o trabalho desta dissertação, suas contribuições e trabalhos futuros.

6.1. Contribuições

A principal contribuição desta dissertação foi apresentar uma arquitetura que busca solucionar o problema de execução de cargas contínuas nos Data Warehouses se apoiando em técnicas de fragmentação e distribuição de dados, já utilizadas tradicionalmente no cenário de Data Warehouse, porém buscando também o desempenho das inserções, e não apenas das consultas. Através do uso destas técnicas, este trabalho permite a carga de dados em tempo real, sem gerar impacto nas ferramentas de BI que acessam dados históricos ou dados recentes. Estes dados são acessados pelas ferramentas sem que as mesmas necessitem conhecer a sua localidade, evitando os problemas que isso acarreta, como, por exemplo, o aumento

de complexidade das consultas. Testes experimentais realizados com o benchmark SSB utilizando um Scale Factor de 100, o que representa uma massa de dados de 100 GB, demonstraram a eficiência da proposta. Os resultados obtidos com a execução das avaliações confirmam a capacidade de execução de cargas contínuas durante consultas, apresentando os seguintes ganhos:

- Consultas executadas simultaneamente às cargas contínuas apresentam desempenho superior em 35,6% na fragmentação proposta em relação à fragmentação tradicional.
- Cargas executadas em conjunto com consultas ocorrem 13 vezes mais rápidas na fragmentação proposta em relação à fragmentação tradicional.
- Tempo para realizar consultas é praticamente o mesmo durante ou na ausência de cargas simultâneas.
- Queda de desempenho das consultas e das cargas após crescimento do fragmento de tempo real ocorre de forma suave, permitindo planejamento de redistribuições.
- Janelas de redistribuições diárias com tempos de indisponibilidades aceitáveis.

Adicionalmente, uma extensão do Star Schema Benchmark foi realizada para permitir a simulação do impacto das cargas contínuas em diversas arquiteturas de DW. Esta extensão produziu modificações no gerador de massa de dados e um novo grupo de consultas para simular o acesso a dados recentes. A partir deste benchmark foi especificado um cenário de experimentação para validação da arquitetura proposta.

6.2. Trabalhos Futuros

Diversos trabalhos ainda podem ser desenvolvidos a fim de continuar a pesquisa elaborada nesta proposta.

Com relação à distribuição e fragmentação, os seguintes cenários podem ser desenvolvidos e comparados a fim de verificar possíveis benefícios e/ou perdas:

- Alocação de mais de um fragmento de dados por nó, permitindo verificar se nós que possuem simultaneamente fragmentos históricos e de tempo real teriam um desempenho melhor que nós especializados em um tipo de fragmento.
- Verificar o desempenho de um ambiente que possua mais de um nó para inserção, avaliando o equilíbrio entre tempo requerido para inserções em tempo real contra tempo das consultas históricas.

- Avaliar como a arquitetura proposta se comporta em ambientes com menor quantidade de nós ou em ambientes com maior quantidade de nós.
- Criar novo fragmento de tempo real quando o desempenho do anterior se degradar, evitando a tarefa de redistribuição dos dados. Neste caso, é necessário avaliar o tempo de criação dos índices e das visões materializadas durante a fase de conversão deste fragmento de tempo real em fragmento histórico para verificar a viabilidade.

Outro ponto envolve estudar a replicação dos fragmentos a fim de permitir alta disponibilidade, avaliando o comportamento da proposta.

Avaliar o impacto que esta proposta causaria em Data Warehouses reais também é um trabalho viável, já que toda a avaliação presente nesta dissertação foi baseada nas simulações de um *benchmark*.

Realizar estudo para avaliar o impacto nas inserções, caso o fragmento de tempo real tivesse índices ou visões materializadas, assim como os possíveis ganhos nas consultas que acessam esses dados.

Adaptar propostas existentes de Real Time Data Warehousing para um cenário distribuído, buscando ou agregar ambas as propostas ou realizar testes comparativos conforme os realizados contra Data Warehouses Tradicionais.

Por fim, buscar explicar a oscilação dos tempos de execução das cargas durante a avaliação experimental, as quais, apesar de não afetarem a viabilidade da proposta, apresentaram um comportamento não linear o qual não foi previsto inicialmente.

Referências

CERI, S., PELAGATTI, G., 1984, *Distributed Databases: Principles and Systems.* New York, McGraw-Hill.

CGOLAP, 2010, *Implementação e Análise de Desempenho de Clusters de PCs Usando Aplicações OLAP sobre Bancos de Dados Espaciais.* Disponível em: <http://www.uniriotec.br/~cgolap/index.html>. Acesso em: 20 mar. 2010.

CHAUDHURI, S., DAYAL, U., 1997, "An Overview of Data Warehousing and OLAP Technology". *ACM SIGMOD Record*, v. 26, n. 1, pp. 65-74.

CHAUDHURI, S., DAYAL, U., GANTI, V., 2001, "Database technology for decision support systems". *Computer*, v. 34, n. 12, pp. 48-55.

DAYAL, U., CASTELLANOS, M., SIMITSIS, A., WILKINSON, K., 2009, "Data integration flows for business intelligence". In: *Proceedings of the 12th International Conference on Extending Database Technology: Advances in Database Technology*, pp. 1-11, Saint-Petersburg, Mar.

DOKA, K., TSOUMAKOS, D., KOZIRIS, N., 2010, "Efficient updates for a shared nothing analytics platform". In: *Proceedings of the 2010 Workshop on Massive Data Analytics on the Cloud*, pp. 7:1-7:6, North Carolina, Apr.

ELNIKETY, S., 2009, "Distributed DBMS". In: Liu, L., Özsu, M.T. (eds), *Encyclopedia of Database Systems,* Springer, pp. 896-899.

FURTADO, P., 2004, "Experimental Evidence on Partitioning in Parallel Data Warehouses". In: *Proceedings of the ACM Seventh International Workshop on Data Warehousing and OLAP*, pp. 23-30, Washington D.C., Nov.

HATAMI, M., 2007, "SOA: Providing Enterprise-Wide Information Access". *InfoManagement Direct*, Apr. 13.

HAYES, A., BROOKS III, E. D., NASH, T., WINKLER, K. H., 1992, "The Role of Computational Clusters". *Proceedings of the 1992 ACM/IEEE conference on Supercomputing*, Minnesota, Nov.

HELLAND, P., 2009, "Process Structure of a DBMS". In: Liu, L., Özsu, M.T. (eds), *Encyclopedia of Database Systems,* Springer, pp. 2178-2187.

HOGAN, M., 2009, *Shared-Disk vs. Shared-Nothing Comparing Architectures for Clustered Databases*. ScaleDB White Paper, Disponível em: <http://www.scaledb.com/pdfs/WP_SDvSN.pdf>.

INMON, W.; STRAUSS, D.; NEUSHLOSS, G., 2008, *DW 2.0 - Architecture for the Next Generation of Data Warehousing*. San Francisco, Morgan Kauffman.

JIMÉNEZ-PERIS, R., PATIÑO-MARTÍNEZ, M., 2009, “Replication for Scalability”. In: Liu, L., Özsu, M.T. (eds), *Encyclopedia of Database Systems,* Springer, pp. 2403-2408.

JOHNSON, T., 2009, “Indexing of Data Warehouses”. In: Liu, L., Özsu, M.T. (eds), *Encyclopedia of Database Systems,* Springer, pp. 1454-1457.

KIMBALL, R., 1998, *Data Warehouse Toolkit*. São Paulo, Makron Books.

KIMBALL, R., CASERTA, J., 2004, *The Data Warehouse ETL Toolkit: Practical Techniques for Extracting, Cleanin*. New York, John Wiley & Sons Inc.

LIMA, A. A. B.; FURTADO, C.; MATTOSO, M.; VALDURIEZ, P., 2009, “Parallel OLAP Query Processing in Database Clusters with Data Replication”. *Distributed and Parallel Databases*, v.25, n. 1-2 (Apr), pp. 97-123.

MARTINS, D. B., 2009, *Extração Personalizada e Incremental de Dados em Ambientes de BI Tempo Real*. Dissertação de Mestrado, DIA/UNIRIO, Rio de Janeiro, RJ, Brasil.

MATTOSO, M., SILVA, G.Z., LIMA, A.A.B., BAIÃO, F.A., BRAGANHOLO, V.P., AVELEDA, A., MIRANDA, B., ALMENTERO, B.K., COSTA, M.N., 2006, “ParGRES: middleware para processamento paralelo consultas OLAP em clusters de Banco de dados”. In: *Proceedings of the 21st Brazilian Symposium on Databases 2nd Demo Session*, pp. 19–24, Florianopolis, Oct.

O'NEIL, P.; O'NEIL, E.; CHEN, X., REVILAK, S., 2009, “The Star Schema Benchmark and Augmented Fact Table Indexing”. In: Nambiar, R., Poess, M. (eds), *Performance Evaluation and Benchmarking*, pp. 237-252, Berlin, Springer-Verlag.

ÖZSU, M. T.; VALDURIEZ, P., 1999, *Principles of Distributed Database Systems*. 2nd ed., New Jersey, Prentice Hall.

PAES, M.; LIMA, A. A. B.; MATTOSO, M., 2009, “Processamento de Alto Desempenho em Consultas sobre Bases de Dados Geoestatísticos Usando Replicação Parcial”. In: *Proceedings of the 24st Brazilian Symposium on Databases*, pp. 241-255, Ceará, Oct.

PEREIRA, D.; AZEVEDO, L. G.; TANAKA, A., 2011, “Real Time Loading of Enterprise Data Using Fragmentation of Data Warehouses”. In: *Proceedings of the 26th Brazilian Symposium on Databases, pp.* 65-72, Florianópolis, Oct.

PGPOOL-II, 2009, *What is pgpool-II?* Disponível em: <http://pgpool.projects.postgresql.org/>, Acesso em: 17 out. 2009.

PGPOOL-II, 2011, *pgpool-II manual*. Disponível em: <http://pgpool.projects.postgresql.org/pgpool-II/doc/pgpool-en.html>, Acesso em: 08 abr. 2011.

POSTGRESQL, 2010, *PostgreSQL 8.4.8 Documentation: Explain*. Disponível em: <http://www.postgresql.org/docs/8.4/static/sql-explain.html>, Acesso em: 23 nov. 2010.

POSTGRESQL, 2011, *PostgreSQL About.* Disponível em: <http://www.postgresql.org/about/>, Acesso em: 12 jun. 2011.

REAL-TIME-SSB, 2011, *A. Real Time SSB Project.* Disponível em: <http://code.google.com/p/real-time-ssb/>, Acesso em: 01 set. 2011.

RISCH, TORE., 2009, “Distributed Architecture”. In: Liu, L., Özsu, M.T. (eds), *Encyclopedia of Database Systems,* Springer, pp. 875-879.

SANTOS, R. J.; BERNARDINO, J., 2008, “Real-time Data Warehouse Loading Methodology”. In: *Proceedings of the 2008 International Database Engineering & Applications Symposium*, pp. 49-58, Coimbra, Sep.

SANTOS, R. J.; BERNARDINO, J., 2009, “Optimizing Data Warehouse Loading Procedures for Enabling Useful-Time Data Warehousing”. In: *Proceedings of the 2009 International Database Engineering & Applications Symposium*, pp. 292-299, Calabria, Sep.

SHI, J., BAO, Y., LENG, F., YU, G., 2009, “Priority-Based Balance Scheduling in Real-Time Data Warehouse”. In: *Proceedings of the 2009 Ninth International Conference on Hybrid Intelligent Systems - Volume 03*, pp. 301-306, Shenyang, Aug.

STODDER, D., 2007, “Good BI, Cruel World?”. *Network Computing*, v. 18, pp. 56-66.

TAN, K., 2009a, “Distributed Database Design”. In: Liu, L., Özsu, M.T. (eds), *Encyclopedia of Database Systems,* Springer, pp. 890-894.

TAN, K., 2009b, “Distributed Database Systems”. In: Liu, L., Özsu, M.T. (eds), *Encyclopedia of Database Systems,* Springer, pp. 894-896.

THIELE, M., FISCHER, U., LEHNER, W., 2007, “Partition-based workload scheduling in living data warehouse environments”. In: *Proceedings of the ACM tenth international workshop on Data warehousing and OLAP*, pp. 57-64, Lisboa, Nov.

THO, M. N., TJOA, A. M., 2004, "Grid-Based Zero-Latency Data Warehousing for Continouos Data Streams Processing". in: *Proceedings of the 6th International Conference on Information Integration and Web-based Applications & Services*, pp. 777 - 786, Jakarta, Sep.

THOMSEN, C.; PEDERSEN, T.; LEHNER, W., 2008, “RiTE: Providing On-Demand Data for Right-Time Data Warehousing”. In: *Proceedings of the 2008 IEEE 24th International Conference on Data Engineering*, pp. 456-465, Cancún, Apr.

TPC-H, 2010, *TPC Benchmark H Revision 2.13.0*. Disponível em: <http://www.tpc.org/tpch/spec/tpch2.13.0.pdf>. Acesso em: 06 abr 2010.

TUNGSTEN., 2010, *Tungsten Replicator Overview*. Disponível em: <http://code.google.com/p/tungsten-replicator/>. Acesso em: 10 jun 2010.

WADA, K., 2009, “Replication”. In: Liu, L., Özsu, M.T. (eds), *Encyclopedia of Database Systems,* Springer, pp. 2391-2392.

WATSON, H.; WIXOM, B., 2007, “The Current State of Business Intelligence”. *Computer*, v. 40, n. 9, pp. 96-99.

ANEXO I – Arquivos de Configuração do pgpool-II

A Figura 68 apresenta os parâmetros de inicialização do pgpool-II, localizados no arquivo pgpool.conf, que foram alterados.

```
replication_mode = true
parallel_mode = true
load_balance_mode = true
enable_query_cache = false

pgpool2_hostname = 'cgolap06'
system_db_hostname = 'cgolap02'
system_db_port = 5446
system_db_dbname = 'pgpool'
system_db_schema = 'pgpool_catalog'
system_db_user = 'pgpool'
system_db_password = ''

backend_hostname0 = 'cgolap01'
backend_port0 = 5446
backend_weight0 = 1
backend_data_directory0 = '/opt/postgres/data/ssb_real'
backend_hostname1 = 'cgolap02'
backend_port1 = 5446
backend_weight1 = 1
backend_data_directory1 = '/opt/postgres/data/ssb_real'
backend_hostname2 = 'cgolap03'
backend_port2 = 5446
backend_weight2 = 1
backend_data_directory2 = '/opt/postgres/data/ssb_real'
```

```
backend_hostname3 = 'cgolap04'
backend_port3 = 5446
backend_weight3 = 1
backend_data_directory3 = '/opt/postgres/data/ssb_real'
backend_hostname4 = 'cgolap05'
backend_port4 = 5446
backend_weight4 = 1
backend_data_directory4 = '/opt/postgres/data/ssb_real'
```

Figura 68 - Valores alterados no pgpool.conf

A Figura 69 apresenta os objetos e tuplas criadas no catálogo do pgpool-II, também chamado de system DB, para realizar a distribuição proposta.

```
CREATE SEQUENCE pgpool_catalog.SSB_node;

CREATE OR REPLACE FUNCTION
pgpool_catalog.dist_def_lineorder_real(anyelement)
RETURNS integer AS $$
SELECT CASE WHEN $1 >= 19990101 THEN 0 ELSE
cast((mod(nextval('pgpool_catalog.SSB_node'),4)+1) as int)
END;

INSERT INTO pgpool_catalog.dist_def
VALUES ( 'postgres', 'public', 'lineorder',
'lo_commitdate', ARRAY ['lo_orderkey', 'lo_linenumber',
'lo_custkey', 'lo_partkey', 'lo_suppkey', 'lo_orderdate',
'lo_orderpriority', 'lo_shippriority', 'lo_quantity',
'lo_extendedprice', 'lo_ordtotalprice', 'lo_discount',
'lo_revenue', lo_supplycost', 'lo_tax', 'lo_commitdate',
'lo_shipmode'], ARRAY ['numeric', 'numeric', 'numeric',
'numeric', 'numeric', 'numeric', 'character(15)',
'character(1)', 'numeric', 'numeric', 'numeric',
'numeric', 'numeric', 'numeric', 'numeric', 'numeric',
'character(10)'],'pgpool_catalog.dist_def_lineorder_real')
```

```
INSERT INTO pgpool_catalog.replicate_def
VALUES ( 'postgres', 'public', 'customer', ARRAY
['c_custkey', 'c_name', 'c_address', 'c_city', 'c_nation',
'c_region', 'c_phone', 'c_mktsegment'], ARRAY ['numeric',
'character(25)', 'character(25)', 'character(10)',
'character(15)', 'character(12)', 'character(15)',
'character(10)'] )

INSERT INTO pgpool_catalog.replicate_def
VALUES ( 'postgres', 'public', 'date', ARRAY ['d_datekey',
'd_date', 'd_dayofweek', 'd_month', 'd_year',
'd_yearmonthnum', 'd_yearmonth', 'd_daynuminweek',
'd_daynuminmonth', 'd_daynuminyear', 'd_monthnuminyear',
'd_weeknuminyear', 'd_sellingseason', 'd_lastdayinweekfl',
'd_lastdayinmonthfl', 'd_holidayfl', 'd_weekdayfl'], ARRAY
['numeric', 'character(18)', 'character(9)',
'character(9)', 'numeric', 'numeric', 'character(7)',
'numeric', 'numeric', 'numeric', 'numeric', 'numeric',
'character(12)', 'boolean', 'boolean', 'boolean',
'boolean'] );

INSERT INTO pgpool_catalog.replicate_def
VALUES ( 'postgres', 'public',  'part', ARRAY
['p_partkey', 'p_name', 'p_mfgr', 'p_category',
'p_brand1', 'p_color', 'p_type', 'p_size', 'p_container'],
ARRAY ['numeric', 'character(22)', 'character(6)',
'character(7)', 'character(9)', 'character(11)',
'character(25)', 'numeric', 'character(10)'] );

INSERT INTO pgpool_catalog.replicate_def
VALUES ( 'postgres', 'public',  'supplier', ARRAY
['s_suppkey', 's_name', 's_address', 's_city', 's_nation',
's_region', 's_phone'], ARRAY ['numeric', 'character(25)',
'character(25)', 'character(10)', 'character(15)',
'character(12)', 'character(15)'] );
```

Figura 69 - objetos e tuplas criadas no system DB para realizar a distribuição proposta