T-SVM: uma abordagem para manutenção de visões materializadas em ambientes *data warehouse*

Luciane Telinski Wiedermann Agner[1]

Universidade Estadual do Centro-Oeste - UNICENTRO
Departamento de Ciência da Computação
Guarapuava, PR, 85015-430, CP: 730

(*Recebido: 10 de dezembro de 2004*)

Resumo: *Data warehouse é um repositório de dados coletados de fontes de dados autônomas e heterogêneas. A tecnologia* data warehousing *tem sido utilizada em Sistemas de Suporte à Decisão (DSS -* Decision Support Systems*) para auxiliar nos processos decisórios. O data warehouse armazena uma ou mais visões materializadas dos dados das fontes. A qualidade do processo de tomada de decisão em um DSS depende de um importante problema, a correta propagação das atualizações ocorridas nas fontes de dados para as visões materializadas no* data warehouse*. Este problema está relacionado com os algoritmos de manutenção de visões que são o foco deste trabalho. Este artigo apresenta diversos algoritmos de manutenção de visões materializadas em ambientes* data warehousing *e propõe um novo algoritmo. O algoritmo proposto, chamado Transaction-SVM (T-SVM), está fortemente baseado no algoritmo SVM. Sua principal vantagem é definir intervalos de tempo para propagar as atualizações no* data warehouse*. O T-SVM adapta o algoritmo SVM para prover consistência forte em cenários de* transação local-fonte.

Palavras-chave: *Data Warehouse, Manutenção de Visões*

Abstract: *A data warehouse is a repository of data collected from different and heterogeneous sources. Data warehouses have been used in Decision Support Systems (DSS) to aim at enabling executives to make better and fast decisions. The data warehouse stores one or more materialized views of the source data. The quality of decisions in a DSS depends on an important issue, that is, the correct propagation of updates at the sources to the views at the data warehouse. This issue is related to view maintenance algorithms that are the*

[1]**E-mail: lagner@unicentro.br**

focus of this work. This paper presents an overview of these algorithms and proposes a new view maintenance algorithm to a data warehouse environment. The proposed algorithm, called Transaction-SVM (T-SVM), is based on SVM algorithm. Its advantage is to define time intervals to propagate the updates at the data warehouse. The T-SVM adapts the SVM algorithm to provide strong consistency for source-local transactions.
Key words: *Data Warehouse, Views Maintenance*

1 Introdução

Data warehouse é um repositório de informações integradas, coletadas de fontes de dados distribuídas, autônomas e, possivelmente, heterogêneas. Os dados armazenados no *data warehouse* são, em geral, utilizados em consultas e análises realizadas por sistemas de suporte à decisão (DSS - *Decision Support Systems*) e *data mining* [26]. Os *data warehouses* visam auxiliar nos processos de tomada de decisão e identificar tendências de mercado [6].

O *data warehouse* armazena uma ou mais visões materializadas sobre os dados das fontes. Uma visão materializada é uma relação derivada, definida em termos das relações base das fontes, armazenada na base de dados do *data warehouse* [9].

Um processo decisório eficaz depende da qualidade e consistência dos dados armazenados no *data warehouse*. Dessa forma, é muito importante realizar a manutenção das visões materializadas do *data warehouse* que visa a refletir as atualizações ocorridas sobre os dados das relações base armazenadas nas diversas fontes de dados. Um dos principais desafios encontrados neste processo é manter a consistência dos dados do *data warehouse*, especialmente quando as fontes de dados são autônomas e as visões materializam dados de múltiplas fontes [27].

Diversas técnicas oferecem soluções para a manutenção das visões materializadas em ambientes *warehousing* [26]:

1) **Reprocessamento**: todos os dados do *data warehouse* são reprocessados periodicamente ou por demanda. Essa técnica consome muito tempo e recursos, particularmente em um ambiente distribuído no qual, geralmente, um grande volume de informações é transferido das fontes para o *data warehouse*. O reprocessamento requer que as consultas sejam interrompidas no *data warehouse* enquanto a atualização da visão materializada está sendo realizada;

2) **Armazenar no** *data warehouse* **cópias dos dados de todas as relações base envolvidas com a visão materializada (replicação)**: nesse caso, os requisitos de armazenamento são muito altos e os dados replicados armazenados no *data warehouse* precisam ser atualizados sempre que ocorre uma mudança nos dados das fontes. Dessa forma, é preciso integrar as atualizações na visão materializada para todos os dados das fontes e não somente para as informações de interesse do sistema de *data warehouse*;

3) **Manutenção incremental**: utiliza algoritmos que processam as atualizações na visão materializada em resposta às alterações ocorridas nas relações base, chama-

dos algoritmos de manutenção incremental de visões materializadas. Somente uma parte da visão materializada é modificada em resposta às atualizações ocorridas nas relações base. Manutenção incremental significa que ao ocorrem mudanças nos dados das fontes, novas informações são coletadas das outras fontes de dados relacionadas e integradas no *data warehouse*. Não é necessário interromper o processamento das consultas realizadas pelas aplicações do cliente ao *data warehouse* enquanto a manutenção é realizada, e, na maioria dos casos, o tempo requerido para a atualização da visão é menor, se comparado à técnica de reprocessamento. Os benefícios da manutenção incremental sobre a técnica de replicação dos dados das fontes são o menor espaço de armazenamento requerido para o *data warehouse* e o menor número de atualizações a serem processadas no *data warehouse* [9]. A principal desvantagem da manutenção incremental é que esta técnica pode gerar uma anomalia nos dados do *data warehouse*.

A última técnica mencionada é o foco deste artigo. Os principais algoritmos de manutenção incremental de visões materializadas em ambientes *data warehousing* são apresentados e um novo algoritmo que combina as principais características destes algoritmos é proposto. O algoritmo proposto, chamado Transaction-SVM (T-SVM), permite ao usuário definir a periodicidade da manutenção, atende a um ambiente de *data warehouse* com múltiplas fontes e proporciona *consistência forte* em cenários de transação local-fonte.

O restante do artigo está organizado como segue. Na Seção 2, são descritas as características e considerações do modelo de *data warehouse* adotado e utilizado ao longo deste artigo. A Seção 3 apresenta um resumo de diversos algoritmos para manutenção incremental de visões materializadas em ambientes *warehousing* e que motivaram o desenvolvimento deste trabalho. A Seção 4 descreve o algoritmo Transaction-SVM, proposto para manutenção incremental em ambientes *warehousing*. Na Seção 5, é apresentado um comparativo entre o algoritmo proposto e os demais algoritmos de manutenção incremental citados na Seção 3. A Seção 6 apresenta as principais contribuições do algoritmo Transaction-SVM e aponta algumas direções para possíveis trabalhos futuros.

2 *Data warehouse*: arquitetura e conceitos relacionados

A arquitetura básica de um ambiente *data warehouse* é apresentada na Figura 1. Nessa arquitetura, o monitor é responsável por identificar as mudanças ocorridas nos dados das fontes e notificar o *data warehouse* e o integrador deve receber os dados das fontes, integrá-los e manter as visões materializadas.

Esta arquitetura foi implementada no projeto WHIPS - *WareHouse Information Prototype at Stanford* [25]. No escopo desse projeto, Zhuge *et al.* [27] fazem algumas considerações e definem alguns aspectos sobre um ambiente *data warehousing*, utilizados ao longo deste trabalho e descritos a seguir. Outro modelo de arquitetura de *data warehouse* foi definido por Devlin [6]. Outro modelo de arquitetura de *data warehouse* foi definido por Devlin [6]. O modelo proposto por Devlin, denominado

Business Data Warehouse (BDW), é baseado no conceito de camada de dados, onde são definidas três camadas distintas: 1) Camada dos Dados Derivados; 2) Camada dos Dados Padronizados e 3) Camada dos Dados de Tempo-Real. O BDW armazena três instantes de tempos para cada registro do *data warehouse*.

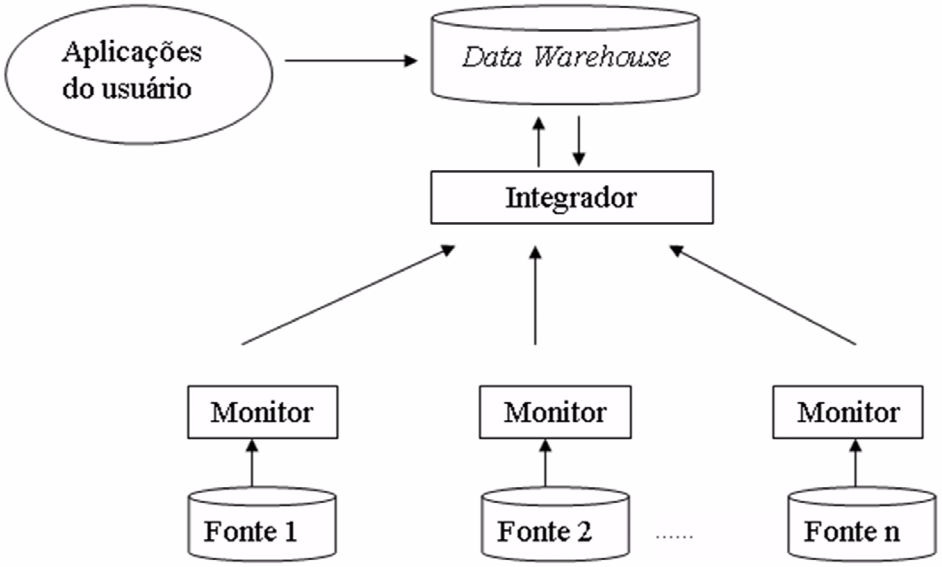

Figura 1 - Arquitetura básica de um *data warehouse*

Assume-se que o integrador é fortemente ligado ao *data warehouse*. Portanto, no restante do trabalho, o termo *warehouse* será utilizado para denotar a combinação entre o integrador e o *data warehouse*.

Os cenários de transação foram definidos com o objetivo de classificar as atualizações que ocorrem nas fontes de dados, sendo eles:

1) **Transações de atualização única**: cada atualização compreende uma transação e é reportada ao *warehouse* separadamente;

2) **Transações local-fonte**: seqüência de atualizações realizadas em uma mesma fonte de dados que juntas compreendem uma transação;

3) **Transações globais**: seqüência de atualizações realizadas em diversas fontes de dados que juntas compreendem uma transação.

Para o propósito deste trabalho, assume-se que as atualizações tratadas pelo *warehouse* são *transações de atualização única e transações local-fonte*. A maioria dos ambientes warehousing suportam cenários de *transações de atualização única, transações local-fonte*, ou ambos [28].

Existem dois tipos de mensagens enviadas das fontes de dados para o *warehouse*: reportar uma transação e retornar a resposta referente a uma consulta. Assume-se

que as *transações de atualização única e as transações local-fonte* são reportadas em uma única mensagem, quando a transação acontece. O *warehouse* envia somente um tipo de mensagem para as fontes de dados: remeter consultas.

Assume-se que a comunicação entre cada fonte de dados e o *warehouse* é confiável e FIFO (*First-in, First-out*), ou seja, mensagens não são perdidas e são entregues na ordem em que foram enviadas. Nenhuma restrição é feita sobre a ordem na qual mensagens enviadas por diferentes fontes são entregues ao *warehouse*, pois assume-se que as fontes de dados são autônomas e as atualizações ocorridas em diferentes fontes não são sincronizadas.

A consistência da manutenção de visões materializadas, para um ambiente *warehousing* com múltiplas fontes de dados, é definida com base nos estados das fontes e do *warehouse*. O estado da fonte é o estado da base de dados da fonte, alterado sempre que a base de dados é atualizada. O estado do *warehouse* é o estado da visão materializada do *warehouse*, modificado sempre que a visão é atualizada. Um algoritmo de manutenção de visões materializadas para ambientes warehousing é responsável por trazer a visão para um novo estado consistente depois de processar as mudanças ocorridas nas fontes, assim, ele gera uma seqüência de estados *warehouse* enquanto está realizando a manutenção da visão materializada.

Os níveis de consistência dependem da forma como as atualizações das fontes são incorporadas na visão materializada do *warehouse*. Cada nível agrupa todos os níveis antecedentes. Os principais níveis de consistência definidos para ambientes *warehousing* são os seguintes:

1) **Convergência**: as mudanças ocorridas nas fontes são eventualmente incorporadas na visão materializada, ou seja, depois da última atualização da visão materializada e quando todas as atividades das fontes tiverem cessado, o estado do *warehouse* é consistente com os estados das fontes;

2) **Consistência Forte**: a ordem dos estados de transformação da visão materializada no *warehouse* corresponde à ordem dos estados de transformação dos dados das fontes. Assim, cada estado do *warehouse* reflete um conjunto de estados das fontes e a ordem dos estados do *warehouse* combina com a ordem dos correspondentes estados das fontes;

3) **Consistência Completa**: cada estado das fontes de dados é refletido em um estado distinto do *warehouse*. Assim, a ordem dos estados de transformação nas fontes de dados é completamente preservada no *warehouse*. Assume-se que a visão materializada do *warehouse* é definida por uma expressão *Project-Select-Join* (PSJ) da álgebra relacional. Assim, se $\{r_1, r_2, ..., r_n\}$ são as n relações base armazenadas nas fontes de dados correspondentes, a visão denotada por V é definida como segue:

$$V = \Pi proj(\sigma cond(r_1 \bowtie r_2 \bowtie ... \bowtie r_n))$$

em que *proj* é um conjunto de nomes de atributos, *cond* é uma expressão condicional, e $r_1, r_2, ..., r_n$, são as relações base. Qualquer expressão da álgebra relacional

construída com as operações *project, select* e *join* pode ser transformada em uma expressão equivalente desta forma [7].

Duas relações base podem residir na mesma fonte de dados ou em fontes distintas. O modelo de dados considerado para as fontes de dados e para o *warehouse* é o modelo de dados relacional.

São considerados dois tipos de atualizações ocorridas nas fontes de dados: inserção e exclusão. Desta forma, a alteração de uma tupla é tratada como uma exclusão da antiga tupla seguida de uma inserção da nova tupla [10]. Quando uma atualização ocorrida na fonte r_i, denominada U_i, é recebida pelo *warehouse*, as mudanças na visão materializada são processadas através de consultas enviadas para as fontes de dados relacionadas. Assim, o *warehouse* gera uma consulta geral, denominada Q, definida como segue:

$$Q = \Pi proj(\sigma cond(r_1 \bowtie ... \bowtie U_i \bowtie ... \bowtie r_n))$$

em que U_i é a tupla t_i atualizada na relação base r_i.

Foram descritos alguns aspectos sobre um ambiente *data warehousing*, utilizados neste trabalho. A seguir, serão apresentados alguns algoritmos de manutenção incremental existentes na literatura, e que serviram de base para o desenvolvimento deste trabalho.

3 Algoritmos de manutenção incremental

Os algoritmos de manutenção incremental da visão materializada atualizam a visão de forma incremental em resposta às atualizações ocorridas nas fontes de dados. Uma revisão sobre manutenção incremental de visões materializadas pode ser encontrada em [9, 10, 13]. Discussões sobre visões materializadas relacionadas com ambientes *warehousing* podem ser encontradas em [11, 12, 16].

Algoritmos de manutenção incremental de visões materializadas foram desenvolvidos para ambientes de base de dados centralizados [3, 4, 5, 8, 9, 14, 18, 22, 23]. A maioria desses algoritmos assume que a manutenção da visão é executada por um sistema único que controla todas as relações base, monitorando as atividades das fontes e definindo quais as informações necessárias para atualizar a visão materializada. Estes algoritmos diferem principalmente quanto ao tipo de definição de visão que eles manipulam. Alguns destes algoritmos controlam as tuplas duplicadas através das informações dos atributos chave das relações base relacionadas com o *warehouse* [4], enquanto outros armazenam como informação adicional na visão materializada o número de derivações de cada tupla [9]. Estas soluções requerem que os dados das relações base sejam bloqueados enquanto as mudanças são processadas na visão.

Um *warehouse* mantém uma cópia sumarizada dos dados das fontes, desta forma, tem-se um ambiente de banco de dados distribuído com dados replicados [27]. Diversos algoritmos estudam a manutenção de visões em ambientes distribuídos [15, 19,

20, 21], mas estas abordagens requerem que as fontes de dados forneçam informações adicionais, como *timestamps*, para atualizar a visão.

Em um ambiente *data warehousing* as fontes podem ser legados ou sistemas não sofisticados, e portanto, possuem as seguintes restrições: 1) Podem informar o *data warehouse* da ocorrência de uma atualização, entretanto, não possuem capacidade de determinar quais os dados pertencentes às fontes de dados relacionadas devem ser integrados com a informação atualizada; 2) Não é possível assumir que elas irão cooperar transmitindo informações adicionais juntamente com as mensagens de atualização.

O artigo focaliza somente as soluções onde as fontes consideradas são autônomas, podendo ser sistemas não sofisticados ou legados. Assim, não são consideradas abordagens onde é preciso bloquear os dados das fontes enquanto o *warehouse* atualiza as visões ou soluções onde as fontes necessitam fornecer informações adicionais para manter as visões.

Diversos algoritmos de manutenção de visões materializadas para ambientes *warehousing* são apresentados na literatura, dentre eles destacam-se: 1) A família de algoritmos *Eager Compensating Algorithm* (ECA) [26]; 2) A família de algoritmos *Strobe* [27]; 3) Os algoritmos SWEEP e Nested SWEEP [2]; e 4) O algoritmo SVM [1].

Os algoritmos ECA [26] consideram um cenário restrito e todos os dados da visão são provenientes de uma única fonte de dados. Os algoritmos da família ECA proporcionam *consistência forte* em modelos de *data warehouse* com uma única fonte e considerando cenários de *transação de atualização única*. A família de algoritmos ECA é formada pelos seguintes algoritmos: 1) Algoritmo ECA (*Eager Compensating Algorithm*); 2) Algoritmo ECA-Key (ECA^K); 3) Algoritmo ECA-Local (ECA^L).

A idéia básica do algoritmo ECA é adicionar *consultas compensadoras* na consulta enviada para a fonte, com o objetivo de compensar os efeitos das atualizações que acontecem de forma concorrente ao processamento de uma consulta. Os algoritmos ECA^K e ECA^L são melhoramentos do algoritmo ECA original. O ECA^K elimina a necessidade de incorporar *consultas compensadoras* nas consultas enviadas para a fonte de dados, utilizando o conceito de atualizações locais, que requer a inclusão dos atributos chave de cada relação na lista de projeção da definição da visão materializada. O ECA^L combina as *consultas compensadoras* do ECA com as atualizações locais do ECA^K.

Os algoritmos da família Strobe [27] foram projetados para cenários de transação que envolvem múltiplas fontes de dados e proporcionam *consistência forte*, ou seja, cada estado do *warehouse* reflete um conjunto de estados consistentes das fontes. Cada algoritmo da família Strobe foi projetado para atender a um cenário de transação especifico: 1) Algoritmo Strobe: projetado para cenários de *transação de atualização única*; 2) Algoritmo Transaction-Strobe: projetado para cenários de *transação local-fonte*; 3) Algoritmo Global-Strobe: projetado para cenários de *transação global*.

Os algoritmos Strobe assumem que a definição da visão inclui na lista de projeção os atributos chave para cada relação base envolvida. Esse requerimento torna estes algoritmos mais restritivos; entretanto, mais eficientes e fáceis de implementar [28].

A idéia principal dos algoritmos Strobe é identificar as atualizações concorrentes que acontecem enquanto determinada consulta é processada. Assim, os efeitos destas atualizações podem ser eliminados do resultado final da consulta. O principal problema dos algoritmos é que a visão materializada não pode ser atualizada até que o sistema de *data warehouse* esteja estável, ou seja, o algoritmo espera até que todas as atualizações cessem e somente então as respostas resultantes de todas as atualizações ocorridas nas fontes são incorporadas na visão materializada. A visão materializada nunca será atualizada se não houver um período de estabilidade no sistema.

Zhuge *et al.* [27] propuseram o algoritmo C-Strobe para contornar a necessidade de um período de estabilidade no sistema de *data warehouse* para atualizar a visão materializada. No C-Strobe, cada atualização é avaliada completamente antes de processar as atualizações subseqüentes. O C-Strobe provê *consistência completa*, ou seja, o estado da visão materializada reflete cada estado de transição das fontes. A principal desvantagem do C-Strobe é que o número de mensagens enviadas para processar cada atualização é muito alto.

Os algoritmos da família ECA e da família Strobe foram implementados no protótipo do sistema WHIPS - *WareHouse Information Prototype at Stanford* [25].

Os algoritmos ECA e Strobe realizam o processamento completo de uma consulta antes de executar qualquer compensação. Dessa forma, todas as atualizações que ocorrem durante o tempo em que a consulta está sendo processada são tratadas como concorrentes, mesmo que estas atualizações não interfiram no resultado da consulta. O algoritmo SWEEP [2] compensa os efeitos somente para as atualizações que realmente interferem no resultado da consulta e esta compensação é realizada diretamente no *warehouse,* utilizando os dados disponíveis nas mensagens de atualização enviadas. Este processo é chamado de *correção de erro on-line*.

Em contraste com os algoritmos Strobe, o SWEEP não requer um estado estável do *warehouse* para incorporar as mudanças na visão. No algoritmo SWEEP, o estado do *warehouse* preserva a ordem de entrega das atualizações ocorridas nas fontes, através da ordenação das mensagens de atualização recebidas. As atualizações que ocorrem nas fontes de dados são ordenadas com base na ordem em que estas atualizações são entregues no *warehouse* e a visão materializada é atualizada na ordem destas atualizações. O SWEEP oferece *consistência completa* para cenários de *transação de atualização única*.

O algoritmo Nested SWEEP [2], uma extensão do algoritmo SWEEP, proporciona *consistência forte* para cenários de *transação de atualização única*. O Nested SWEEP processa as mudanças na visão materializada para múltiplas atualizações das fontes coletivamente, compartilhando componentes dos resultados das consultas da atualização que está sendo processada com as atualizações subseqüentes. O Nested SWEEP requer um período em que não ocorram atualizações concorrentes nas fontes de dados para atualizar a visão materializada. Maiores detalhes sobre o algoritmo Nested SWEEP podem ser encontrados em [24].

O algoritmo SVM (*Algorithm for Scheduling Warehouse View Maintenance*) [1] reúne as principais características dos algoritmos Strobe e SWEEP e realiza a manutenção incremental de visões materializadas em ambientes *warehousing* de

forma programada. A proposta do SVM é atualizar a visão materializada periodicamente e incorporar um conjunto de mudanças das fontes a cada atualização. O SVM foi projetado para ambientes *warehousing* com múltiplas fontes de dados e proporciona *consistência forte* em cenários de transação de atualização única, ou seja, cada estado *warehouse* reflete um conjunto de estados das fontes.

4 Algoritmo T-SVM

Esta seção descreve o algoritmo Transaction-SVM (T-SVM) proposto para manutenção incremental de visões materializadas em ambientes *data warehouse* e define seu contexto de aplicação.

O modelo de *data warehouse*, apresentado na Seção 2, foi adotado como base para desenvolvimento do algoritmo T-SVM. Cada fonte de dados pode armazenar qualquer número de relações base, mas para simplificar a descrição do algoritmo assume-se que cada fonte possui somente uma relação.

O algoritmo T-SVM adapta o algoritmo SVM [1] para prover *consistência forte* em cenários de *transação local-fonte*. O T-SVM coleta todas as atualizações executadas por uma transação e processa estas atualizações como uma unidade única. Agrupar as atualizações de uma transação reduz o número de mensagens enviadas e facilita o processo de atualização do *warehouse*. O algoritmo T-SVM também utiliza os principais fundamentos dos algoritmos Strobe e SWEEP.

O algoritmo T-SVM, apresentado na Figura 2, possui dois módulos: o monitor do *warehouse* e o monitor da fonte de dados. A Tabela 1 apresenta a definição dos termos que ilustram os conceitos, variáveis e funções utilizados na descrição do algoritmo proposto. Muitos desses termos encontram-se definidos em [2, 27]. Cada fonte de dados pode armazenar qualquer número de relações base, mas para simplificar a descrição do algoritmo assume-se que cada fonte possui somente uma relação. O monitor da fonte de dados deve agrupar, em uma única transação, diversas atualizações ocorridas em determinado espaço de tempo em sua base de dados. Então, o monitor da fonte deve enviar a transação ocorrida ao *data warehouse* através do envio de uma mensagem de aviso de atualização. Outra função do monitor da fonte é processar consultas enviadas pelo *data warehouse*, e retornar o resultado do processamento destas consultas ao *warehouse*.

Ao receber uma mensagem de aviso de atualização de uma das fontes de dados relacionadas, o monitor do *warehouse* deve remover da transação todos os pares de inserção e exclusão de uma mesma tupla, existentes na transação a ser processada e definidas nesta ordem: primeiro acontece a inserção e depois a exclusão da tupla. Esta característica evita o envio de consultas para processar a inserção de uma tupla que posteriormente será excluída. Em seguida, o monitor deve incluir na fila de mensagens *Message Queue* (MQ) e em SUMQ(i) toda atualização existente na transação. As mensagens existentes em MQ são processadas seqüencialmente.

Quando a mensagem refere-se a uma exclusão, o algoritmo adiciona uma ação de exclusão na lista de ações (AL). Para mensagens de inclusão, o monitor do *warehouse*

deve gerar uma consulta a ser avaliada pelas fontes de dados relacionadas.

Algoritmo T-SVM:
Monitor da fonte de dados:
- Agrupa toda as atualizações U_i, ocorridas em determinado espaço de tempo, na transação T_i.
- Envia T_i para o *data warehouse*.
- Quando recebe uma consulta Q_i, processa a consulta Q_i e envia a resposta A_i ao *data warehouse*.

Monitor do *Warehouse*:
Recebe Mensagens:
- Quando recebe uma transação T_i da fonte i:
  - Exclui $\forall U_i \in T_i$ referente a pares de inserção e remoção da mesma tupla, ocorridas nesta ordem, existentes em T_i.
  - Adiciona $\forall U_i \in T_i$ na fila MQ e em SUMQ(i).

Processa Mensagens:
- Para $\forall Ui \in MQ$, retira U_i de MQ e de SUMQ(i). Processa cada atualização U_i existente em MQ conforme ordem de chegada.
- Se U_i for uma exclusão
  - AL = AL + delete(MV,U_i).
- Se U_i for uma inserção
  - Seja $Q_i = Q<U_i>$, gera Q_i com base nas informações da atualização U_i.
  - Para cada fonte j localizada à esquerda da fonte i
    - $Q_i' = ParseQuery(Q_i, j)$.
    - Envia Q_i' para a fonte j.
    - Após receber a resposta A_i' da fonte j
      - Se $\exists U_k \in SUMQ(j)$, então
        - $\forall U_k$, tal que U_k é uma inserção, faça $(A_i' = A_i' - U_k)$.
        - $\forall U_k$, tal que U_k é uma exclusão, faça $(A_i' = A_i' + U_k)$.
      - $A_i = A_i + A_i'$.
  - Para cada fonte j localizada à direita da fonte i
    - $Q_i' = ParseQuery(Q_i, j)$.
    - Envia Q_i' para a fonte j.
    - Após receber a resposta A_i' da fonte j:
      - Se $\exists U_k \in SUMQ(j)$, então
        - $\forall U_k$, tal que U_k é uma inserção, faça $(A_i' = A_i' - U_k)$.
        - $\forall U_k$, tal que U_k é uma exclusão, faça $(A_i' = A_i' + U_k)$.
    - $A_i = A_i + A_i'$.
  - AL = AL + insert(MV, A_i).

Atualiza o Data Warehouse:
- AL é inicializada como uma lista vazia, AL = < >.
- Se *CountTime*( ) for igual a TIME, (*CountTime* = TIME), faça
- MV = MV + AL, aplica as ações de AL em MV
- Define AL = < >.

fim do algoritmo.

Figura 2 - Algortimo T-SVM.

As consultas geradas pelas mensagens de inclusão são processadas seqüencialmente. Em um ambiente distribuído, a consulta é dividida em porções e enviada para as fontes de dados relacionadas. Para exemplificar a ordem de processamento

das consultas, assume-se que a visão materializada do *warehouse* é definida através da seguinte expressão:

$$V = (r_1 \bowtie \quad \ldots \quad \bowtie r_{i-1} \bowtie r_i \bowtie r_{i+1} \bowtie \quad \ldots \quad \bowtie r_n)$$

onde r_1, ..., r_{i-1}, r_i, r_{i+1},..., r_n são as n relações base armazenadas nas fontes de dados. Supondo que uma atualização U_i da relação ri é recebida no *warehouse*. As mudanças são processadas através de consultas enviadas para as fontes de dados relacionadas. Assim, o *warehouse* gera uma consulta geral, denominada Q_i, definida como segue:

$$Q_i = (r_1 \bowtie \quad \ldots \quad \bowtie r_{i-1} \bowtie U_i \bowtie r_{i+1} \bowtie \quad \ldots \quad \bowtie r_n)$$

Termo	**Definição**
A_i	Resposta referente a consulta Q_i.
AL	Lista de ações a serem aplicadas no *warehouse*.
CountTime( )	Função que controla a ocorrência de um intervalo de tempo.
insert(MV, U_i)	Função que insere a tupla U_i na visão materializada.
delete(MV, U_i)	Função que exclui a tupla U_i na visão materializada.
MQ	*Message Queue*, fila que armazena o conjunto de mensagens de atualização recebidas pelo *warehouse*.
MV	Visão materializada armazenada no *warehouse*.
ParseQuery(Q_i,j)	Função que retorna a porção Q_i^j da consulta Q_i, a ser avaliada pela fonte j.
Q_i	Consulta gerada devido a atualização U_i ocorrida em uma das fontes de dados relacionadas.
$Q<U_i>$	$Q<U_i>$ - denota a expressão da definição da visão V, com a relação base onde ocorreu U_i substituída pela tupla referente à U_i.
TIME	Variável que armazena o valor do intervalo de tempo estabelecido para atualizar a visão materializada.
SUMQ(i)	*Source Update Message Queue*, conjunto de mensagens de atualização ocorridas na fonte i. O conjunto SUMQ(i) é um subconjunto do conjunto MQ (SUMQ(i) $\subset$ MQ).
U_i	Atualização ocorrida em uma das fontes de dados relacionadas com o *warehouse*.
T_i	Transação que agrupa diversas atualizações ocorridas em uma mesma fontes de dados.

Tabela 1 - Terminologia utilizada no algoritmo SVM

O algoritmo T-SVM envia primeiramente as porções da consulta Q_i para as relações na direção esquerda da relação r_i onde ocorreu a atualização U_i, considerando a expressão condicional da definição da visão. Depois de enviar a consulta

para todas as relações da esquerda, a consulta é enviada para as relações na direção direita de r_i. A consulta é processada iterativamente, como segue: primeiro na direção esquerda de r_i e então prossegue na direção direita de r_i. Desta forma, consulta é primeiramente avaliada em r_{i-1}, depois em r_{i-2}, e assim por diante. O *warehouse* envia a consulta para ser processada em r_{i-1}, e depois de receber a resposta A_{i-1}, continua o processamento enviando a consulta para r_{i-2}. O processamento da consulta acontece da mesma forma na direção direita. A Figura 3 representa a ordem de processamento da consulta Q_i.

Desta forma, assume-se uma topologia virtual das fontes de dados, definida através da expressão condicional da definição da visão, utilizada para estabelecer a ordem de envio das consultas para as fontes de dados. Esta forma de processamento das atualizações foi definida com base na *fila de mensagens de atualização* proposta no algoritmo SWEEP [2].

Enquanto uma consulta está sendo processada pelas fontes de dados, novas atualizações podem ocorrer. Para eliminar o efeito das atualizações concorrentes, o monitor do *warehouse* ao receber a resposta de uma fonte de dados, referente à porção de uma consulta avaliada, verifica se não ocorreram novas atualizações nesta fonte do instante em que a consulta foi gerada até o momento em que a consulta foi processada pela fonte. Para verificar a existência de atualizações concorrentes é utilizado o conjunto SUMQ(j) que guarda as atualizações ocorridas na fonte de dados j e que ainda não foram processadas pelo *warehouse*. A compensação é realizada localmente no *warehouse* através das informações existentes na fila de mensagens e em SUMQ(j), sem a necessidade de gerar e enviar novas consultas para as fontes de dados relacionadas.

Os resultados das consultas e as ações de exclusão não são aplicados diretamente no *warehouse*. Em vez disso, o algoritmo armazena todas as ações a serem executadas na visão materializada na lista de ações. Desta forma, é possível realizar uma manutenção programada da visão materializada, através da definição de intervalos de tempo para aplicar as ações de AL no *warehouse*.

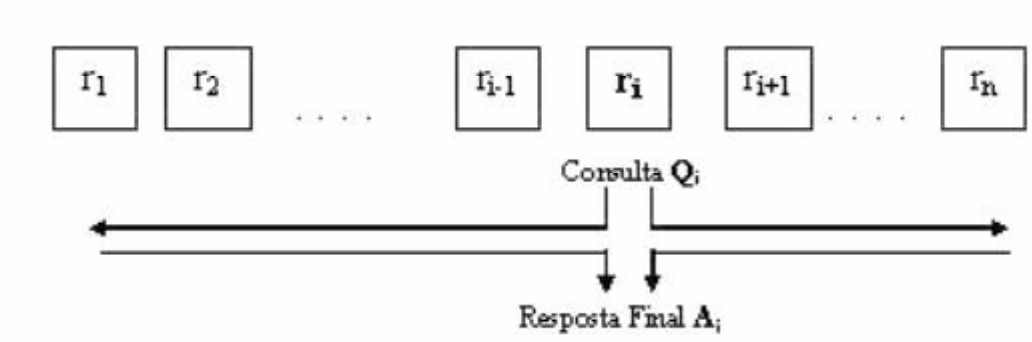

Figura 3 - Processamento iterativo da consulta

Os resultados das consultas e as ações de exclusão não são aplicados diretamente no *warehouse*. Em vez disso, o algoritmo armazena todas as ações a serem executadas na visão materializada na lista de ações. Desta forma, é possível realizar uma manutenção programada da visão materializada, através da definição de intervalos de tempo para aplicar as ações de AL no *warehouse*.

O processo de atualização da visão materializada é realizado periodicamente pela função *CountTime*. A periodicidade da manutenção da visão materializada é definida pelo usuário do sistema de *data warehouse* conforme as especificações e requisitos do sistema. Os processos de recebimento de mensagens, processamento de mensagens e atualização do *data warehouse* são executados em paralelo pelo monitor do *warehouse*.

O algoritmo T-SVM proporciona *consistência forte*, ou seja, o estado do *warehouse* reflete um conjunto de estados consistentes das fontes de dados.

5 Um comparativo entre o algoritmo T-SVM e os demais algoritmos apresentados

A Tabela 2 apresenta um resumo do comparativo das propriedades dos algoritmos de manutenção incremental de visões materializadas para ambientes *warehousing*.

O algoritmo ECA atende a um cenário de *data warehouse* bastante restrito, onde a visão materializada está relacionada com uma única fonte de dados. Desta forma, o algoritmo T-SVM não pode ser comparado diretamente com o algoritmo ECA.

Em um cenário de *data warehouse* mais abrangente, com diversas fontes de dados relacionadas, o processo de manter a consistência dos dados da visão torna-se mais complexo. Os algoritmos Strobe, T-Strobe, C-Strobe, SWEEP, Nested SWEEP, SVM e T-SVM são aplicados em cenários de *data warehouse* onde existem diversas fontes de dados relacionadas e cada atualização ocorrida nas fontes de dados é reportada separadamente ao *warehouse* ou reportada através de uma transação que engloba diversas atualizações ocorridas em uma mesma fonte de dados.

Os algoritmos SWEEP e C-Strobe proporcionam *consistência completa* em cenários de *transação de atualização única*. *Consistência completa* determina que cada estado do *warehouse* deve refletir um estado de transição das fontes de dados, ou seja, cada atualização ocorrida em uma fonte é refletida em um estado distinto da visão materializada. Entretanto, *consistência completa* pode ser um requisito muito forte e desnecessário na maioria dos cenários de *data warehouse*, pois a visão materializada é atualizada para cada mudança ocorrida nas fontes de dados [27]. Os algoritmos Strobe, T-Strobe, Nested SWEEP, SVM e T-SVM proporcionam *consistência forte*, um requisito desejável na maioria dos cenários de *data warehouse* atuais. *Consistência forte* requer que a visão materializada incorpore o efeito de várias atualizações das fontes coletivamente [28].

A vantagem do SWEEP sobre o algoritmo C-Strobe, considerando que ambos oferecem *consistência completa*, é que no SWEEP o número de mensagens envolvidas

no processamento de uma atualização é significativamente menor. No C-Strobe o número total de consultas enviadas para processar uma atualização é de (n-1)! no pior caso, contra (n-1) mensagens necessárias utilizando o SWEEP, onde n é o número de relações base que participam da definição da visão do *warehouse*.

Algoritmo	Número de fontes	Nível de Consistência	Custo*	Cenários de Transação	Visão inclui atributos chave	Manutenção da visão materializada
ECA	1	*forte*	O(1)	*atualização única*	SIM	Requer um estado estável do sistema de *data warehouse*
Strobe	n	*forte*	O(n)	*atualização única*	SIM	Requer um estado estável do sistema de *data warehouse*
T-Strobe	n	*forte*	O(n)	*local-fonte*	SIM	Requer um estado estável do sistema de *data warehouse*
G-Strobe	n	*forte*	O(n)	*global*	SIM	Requer um estado estável do sistema de *data warehouse*
C-Strobe	n	*completa*	O($n!$)	*atualização única*	SIM	Atualiza a visão depois de cada atualização processada
SWEEP	n	*completa*	O(n)	*atualização única*	NÃO	Atualiza a visão depois de cada atualização processada
Nested SWEEP	n	*forte*	O(n)	*atualização única*	NÃO	Requer um período em que cessem as atualizações concorrentes
SVM	n	*forte*	O(n)	*atualização única*	SIM	Manutenção Programada
T-SVM	n	*forte*	O(n)	*local-fonte*	SIM	Manutenção Programada

*Custo de mensagens (atualização)

Tabela 2 - Comparativo dos algoritmos de manutenção de visões materializadas.

A principal vantagem dos algoritmos SVM e T-SVM sobre os outros algoritmos que oferecem *consistência forte*, é a possibilidade de determinar quando a visão materializada do *warehouse* deve ser atualizada. Nos algoritmos Strobe e T-Strobe

a visão materializada é atualizada somente quando não existem consultas sendo processadas, portanto, se ocorrerem seguidas modificações nas fontes de dados o *warehouse* não será atualizado. Para incorporar na visão materializada as mudanças ocorridas nas fontes, o Nested SWEEP requer que não existam atualizações efetivamente concorrentes sendo processadas, o que é um requisito mais suave que o exigido pelo Strobe e T-Strobe. Desta forma, é possível dizer que os algoritmos Strobe, T-Strobe e Nested SWEEP dependem de algum fator determinante, relacionado ao volume de mensagens de atualização recebidas, para incorporar na visão materializada as mudanças ocorridas nas fontes de dados. Utilizando os algoritmos SVM e T-SVM é possível atualizar a visão em períodos de tempo previamente definidos, sem necessitar que o recebimento de mensagens de atualização seja interrompido por um período de tempo no *warehouse*. Outro benefício da manutenção programada é que o usuário do sistema de *data warehouse* é quem determina a periodicidade de manutenção da visão materializada.

Os algoritmos SWEEP, Nested SWEEP, SVM e T-SVM realizam a compensação das anomalias somente para as atualizações efetivamente concorrentes, ou seja, para as atualizações que ocorreram na fonte de dados i antes da consulta que está sendo processada no *warehouse* ser avaliada na fonte i. Isto não acontece com os algoritmos Strobe que avaliam totalmente o resultado de uma consulta antes de executar qualquer compensação. Utilizando os algoritmos Strobe, a compensação é realizada para todas as atualizações que ocorreram durante o período de processamento da consulta e não somente para as atualizações que realmente interferem no resultado da consulta.

Uma restrição dos algoritmos da família Strobe e dos algoritmos SVM e T-SVM é que a definição da visão deve incluir os atributos chave para cada relação base que participa do *warehouse*. Entretanto, esta restrição permite reduzir o número de mensagens necessárias para processar as mudanças na visão, visto que, as operações de exclusão são executadas diretamente na visão materializada do *warehouse*, sem a necessidade de gerar e enviar consultas para as fontes de dados relacionadas. A maioria das aplicações inclui os atributos chave para as relações base, entretanto, para as aplicações onde os atributos chave não fazem parte da lista de projeção da definição da visão é possível adicionar estes atributos através do *warehouse* [27].

Os algoritmos ECA, Strobe, C-Strobe, SWEEP, Nested SWEEP e SVM foram projetados para cenários de *transação de atualização única*. Os algoritmos T-Strobe e T-SVM foram projetados para cenários de *transação local-fonte* e o algoritmo G-Strobe para cenários de *transação global*.

Em cenários de *transação local-fonte*, é possível agrupar diversas atualizações ocorridas em uma fonte de dados em uma única transação a ser processada no *data warehouse*. Agrupar as atualizações de uma transação reduz o número de mensagens de consulta enviadas e otimiza o processo de atualização do *warehouse*.

6 Conclusões

Data warehouse é um repositório de dados coletados de fontes de dados autônomas e heterogêneas. Os dados do *data warehouse* são utilizados para auxiliar nos processos de tomada de decisão e identificar tendências de mercado. O *data warehouse* armazena uma ou mais visões materializadas dos dados das fontes.

A manutenção incremental das visões materializadas em *data warehouses* deve refletir as atualizações ocorridas sobre os dados das fontes. As vantagens da manutenção incremental sobre as técnicas de reprocessamento e replicação dos dados são principalmente o menor tempo requerido para atualizar a visão se comparada à técnica de reprocessamento e o menor espaço de armazenamento requerido com relação à técnica de replicação.

Este artigo apresentou diversos algoritmos de manutenção incremental de visões materializadas para ambientes *warehousing*. Como resultado desse estudo um novo algoritmo foi proposto, o algoritmo T-SVM que reúne os principais fundamentos destes algoritmos. O T-SVM foi projetado para ambientes *warehousing* com múltiplas fontes de dados e proporciona *consistência forte* em cenários de *transação local-fonte*, ou seja, cada estado *warehouse* reflete um conjunto de estados das fontes. *Consistência forte* é um requisito de consistência adequado para a maioria dos cenários de *data warehouse* atuais. Através do T-SVM é possível realizar uma manutenção programada da visão materializada, ou seja, o *data warehouse* é atualizado periodicamente. A periodicidade da manutenção do *warehouse* é definida pelo usuário final.

Uma sugestão para trabalho futuro é a implementação do algoritmo T-SVM, utilizando um caso real, com o objetivo de coletar métricas, como: 1) Desempenho com relação ao número médio de consultas necessárias para processar múltiplas atualizações e 2) Desempenho com relação ao volume de dados transferido.

Referências

[1] L T. W. Agner e S. R. VERGILIO. SVM - Uma Proposta para a Manutenção de Visões Materializadas em Ambientes Data Warehousing. XXVII Conferência Latinoamericana de Informática (CLEI'2001), Setembro 2001.

[2] D. Agrawal, A. E. Abbadi, A. K. Singh, and T. Yurek. Efficient view maintenance at data warehouses. In **Proceedings of the ACM SIGMOD International Conference on Management of Data**, pages 417-427, May 1997.

[3] J. A. Blakeley, P.-A. Larson, and F. W. Tompa. Efficiently updating materialized views. In **Proceedings of the ACM SIGMOD International Conference on Management of Data**, pages 61-71, June 1986.

[4] S. Ceri and J. Widom. Deriving production rules for incremental view maintenance. In **Proceedings of the 17th International Conference on Very**

Large Data Base (VLDB), pages 577-589, September 1991.

[5] L. S. Colby, T. Griffin, L. Libkin, I. S. Mumick, and H. Trickey. Algorithms for deferred view maintenance. In **Proceedings of the ACM SIGMOD International Conference on Management of Data**, pages 469-480, June 1996.

[6] B. Devlin. **Data Warehouse from Architecture to Implementation**. Addison-Wesley, 1997.

[7] R. Elmasri and S. B. Navathe. **Fundamentals of Database Systems**. Addison-Wesley Publishing Company, 1994.

[8] T. Griffin and L. Libkin. Incremental maintenance of views with duplicates. In **Proceedings of the ACM SIGMOD International Conference on Management of Data**, pages 328-339, May 1995.

[9] A. Gupta, I. Mumick, and V. Subrahmanian. Maintaining views incrementally. In **Proceedings of the ACM SIGMOD International Conference on Management of Data**, pages 157-166, May 1993.

[10] A. Gupta and I. Mumick. Maintenance of Materialized Views: Problems, techniques, and applications. **IEEE Bulletin of the Technical Committee on Data Engineering**, 18(2):3-18, June 1995.

[11] A. Gupta and J. A. Blakeley. Using partial information to update materialized views. **Information Systems**, 20(8):641-662, November 1995.

[12] A. Gupta and I. Mumick, editors. **Proceedings of the workshop on materialized views: techniques and applications**, June 1996.

[13] A. Gupta and I. Mumick, editors. **Materialized Views**. MIT Press, 1998.

[14] J. V. Harrison and S. W. Dietrich. Maintenance of materialized views in a deductive database: An update propagation approach. In **Proceedings of the 1992 JICLSP Workshop on Deductive Databases**, pages 56-65, 1992.

[15] B. Lindsay, L. M. Haas, C. Mohan, H. Pirahesh, and P. Wilms. A snapshot differential refresh algorithm. In **Proceedings of the ACM SIGMOD International Conference on Management of Data**, May 1986.

[16] D. Lomet and J. Widom, editors. **Special Issue on Materialized Views and Data Warehousing**, IEEE Data Engineering Bulletin 18(2), June 1995.

[17] A.T. Pozo et al. SAGU - Sistema de Apoio ao Gerenciamento Universitário. Relatório técnico do Projeto, DINF - Departamento de Informática, UFPR, agosto 2000.

[18] X. Qian and G. Wiederhold. Incremental recomputation of active relational expressions. **IEEE Transactions on Knowledge and Data Engineering**, 3(3): 337-341, September 1991.

[19] A. Segev and W. Fang. Currency-based updates to distributed materialized

views. In **Proceedings of the 6th International Conference on Data Engineering (ICDE)**, pages 512-520, February 1990.

[20] A. Segev and W. Fang. Optimal update policies for distributed materialized views. **Management Science**, 37(7):851-70, July 1991.

[21] A. Segev and J. Park. Updating distributed materialized views. In **IEEE Transactions on Knowledge and Data Engineering**, 1(2):173-184, June 1989.

[22] O. Shmueli and A. Itai. Maintenance of views. In **Proceedings of the ACM SIGMOD International Conference on Management of Data**, pages 240-255, May 1984.

[23] M. Staudt and M. Jarke. Incremental maintenance of externally materialized views. In **Proceedings of the 22nd International Conference on Very Large Data Base (VLDB)**, pages 75-86, September 1996.

[24] T. Yurek. Efficient View Maintenance at Data Warehouses. Master thesis, University of California at Santa Barbara, Department of Computer Science, UCSB, Santa Barbara, CA 93106, 1997.

[25] J. L. Wiener, H. Gupta, W. J. Labio, Y. Zhuge, H. Garcia-Molina, and J. Window. A system prototype for warehouse view maintenance. In **Proceedings of the Workshop on Materialized Views, Techniques and Applications**, pages 26-33, June 1996.

[26] Y. Zhuge, H. Garcia-Molina, J. Hammer, and J. Widom. View maintenance in a warehousing enviroment. In **Proceedings of the ACM SIGMOD International Conference on Management of Data**, pages 316-327, May 1995.

[27] Y. Zhuge, H. Garcia-Molina, and J. L. Wiener. Consistency algorithms for multi-source warehouse view maintenance. **Distributed and Parallel Databases**, 6(1): 7-40, January 1998.

[28] Y. Zhuge. Incremental Maintenance of Consistent Data Warehouses. Phd thesis, Stanford University, Department of Computer Science, Stanford, CA 94305-2140, June 1999.