Técnica de Controle de Concorrência Semântico para Sistemas de Gerenciamento de Bancos de Dados em Tempo-Real

Yáskara Ygara Menescal Pinto Fernandes

Dissertação de Mestrado submetida à Coordenação do Programa de Pós-Graduação em Engenharia Elétrica da Universidade Federal de Campina Grande - Campus de Campina Grande como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências no Domínio da Engenharia Elétrica.

Área de Concentração: Processamento da Informação

Angelo Perkusich, Dsc.
Orientador
Maria Lígia Barbosa Perkusich, Dsc.
Orientadora

Campina Grande, Paraíba, Brasil

Técnica de Controle de Concorrência Semântico para Sistemas de Gerenciamento de Bancos de Dados em Tempo-Real

Yáskara Ygara Menescal Pinto Fernandes

Dissertação de Mestrado apresentada em Agosto de 2005

Angelo Perkusich, Dsc.
Orientador
Maria Lígia Barbosa Perkusich, Dsc.
Orientadora

Maria de Fátima Queiroz Vieira, Ph.D.
Componente da Banca
Péricles Rezende Barros, ph.D.
Componente da Banca

Campina Grande, Paraíba, Brasil, Agosto de 2005

FICHA CATALOGRÁFICA ELABORADA PELA BIBLIOTECA CENTRAL DA UFCG

F363t Fernandes, Yáskara Ygara Menescal Pinto
2005 Técnica de Controle de Concorrência Semântico para Sistemas de Gerenciamento de Bancos de Dados em Tempo-Real / Yáskara Ygara Menescal Pinto Fernandes. — Campina Grande, 2005.
77f. : il.

Inclui bibliografia.
Tese (Mestrado em Engenharia Elétrica) – Universidade Federal de Campina Grande, Centro de Engenharia Elétrica e Informática.
Orientador: Angelo Perkusich e Maria Lígia Barbosa Perkusich.

1— Bancos de Dados em Tempo Real 2— Controle de Concorrência 3— Sistemas em Tempo Real I— Título

CDU 004.655.3 + 004.031.43

TÉCNICA DE CONTROLE DE CONCORRÊNCIA SEMÂNTICA PARA SISTEMAS DE GERENCIAMENTO DE BANCOS DE DADOS EM TEMPO-REAL.

YÁSKARA YGARA MENESCAL PINTO FERNANDES

Dissertação Aprovada em 02.09.2005

ANGELO PERKUSICH, D.Sc., UFCG
Orientador

MARIA LÍGIA BARBOSA PERKUSICH, D.Sc., UNICAP
Orientadora (Ausente)

Maria de Fatima Q. Vieira Turnell

MARIA DE FÁTIMA QUEIROZ VIEIRA TURNELL Ph.D., UFCG
Componente da Banca

Péricles Rezende Barros

PÉRICLES REZENDE BARROS, Ph.D., UFCG
Componente da Banca

CAMPINA GRANDE - PB
Setembro - 2005

Dedicatória

Dedico esta dissertação ao meu esposo Pedro, aos meus filhos Yasmin, Yngrid e Pedro Filho.

Agradecimentos

Estas primeiras páginas, e últimas palavras que escrevo neste trabalho é para dedicar a todas as pessoas que me aconselharam, motivaram, orientaram, reforçaram, cuidaram, ouviram, protegeram e colaboraram ao longo desta minha época especial de vida e de trabalho. Para além destas palavras escritas, espero encontrar melhor forma e melhor momento para dizer a todos o quanto estou agradecida e o quanto sinto que, a todos, devo um bocadinho deste trabalho.

Agradeço a DEUS, pela minha existência, porque nada nos é possível se não for de sua vontade e por está presente em todos os momentos da minha vida.

Aos meus orientadores, Professor Dr. Angelo e Professora Dr_a Lígia, pela oportunidade de realizar o mestrado no Departamento de Engenharia de Elétrica, e pela orientação na elaboração desta dissertação.

Aos meus pais, Aurivan e Jara, pessoas que, com o uso de palavras, seria difícil homenageá-los, pois aprendi com eles, que isto se faz com gestos e atitudes. Agradeço a eles, pela formação, apoio em todos os sentidos e, constantes incentivos.

A minha amiga Cícilia, que foi minha companheira no desenvolvimento deste trabalho, e cujo auxílio foi muito importante para a conclusão do mesmo.

Aos meus filhos: Yasmin, Yngrid e Pedro Filho, que com amor compreenderam minha "presença ausente".

Gostaria de fazer dois agradecimentos especiais.

A minha orientadora e amiga Professora Dr_a Lígia, uma pessoa incentivadora e sempre disposta a ajudar, com sugestões e críticas técnicas extremamente valiosas, que enriquecem a base teórica e prática deste trabalho, sempre expostas de uma maneira muito meiga, a qual lhe é peculiar. Professora Lígia, muito obrigada!

E ao meu esposo Pedro, por acreditar e me incentivar na realização deste trabalho e por está sempre ao meu lado nos momentos mais difíceis.

Agradeço a CAPES pelo auxílio concedido.

Resumo

O grande avanço tecnológico na área de rede de sensores tem motivado o uso de tais redes em uma variedade de aplicações, entre elas podemos destacar: sistemas de monitoramento, detecção de enchente, rastreamento dos produtos estocados em um armazém, entre outros. A principal característica dessas aplicações é o grande volume de dados com tempo de vida útil limitado que precisa ser processado, conseqüentemente as transações que os utilizam também devem ter restrições temporais de forma que consigam usar tais dados enquanto os mesmos ainda sejam válidos. Tais características impõem a necessidade de tais aplicações serem gerenciadas através de Sistemas de Gerenciamento de Banco de Dados em Tempo-Real, que além de serem projetados para tratar com grandes volumes de dados, também garantem a consistência lógica e temporal dos dados e transações. Nesse trabalho, apresenta-se uma Técnica de Controle de Concorrência para Sistemas de Gerenciamento de Banco de Dados em Tempo-Real, denominada Técnica de Controle de Concorrência Semântico, que permite a negociação entre a consistência lógica e temporal quando houver conflito entre ambas, e limita a imprecisão resultante da negociação entre as mesmas. Essa técnica é baseada em uma Função de Compatibilidade, definida pelo usuário, para as operações de leitura e escrita de um objeto da aplicação. A Função de Compatibilidade também controla e limita a imprecisão acumulada quando ocorre a negociação entre consistência lógica e temporal. A Técnica de Controle de Concorrência Semântico foi implementada na linguagem de programação Java e o gerenciador utilizado foi o Sistema de Gerenciamento de Banco de Dados DB2 da IBM.

Abstract

The great technological advance in the areas Sensor Networks has motivated the use of such networks a variety of applications, among them can detach: monitoring systems, flood detection system, supervising items in a factory warehouse, among others. The principal characteristic of those applications it is the large volume of data with time of useful life limited that they need to be processed, consequently the transactions that also use them they should have temporal constraints so that they get to use such data while the same ones are still valid. Such characteristics impose the necessity of such applications to be managed through a Real-Time Database System, that besides being projected to deal with great volumes of data, also they guarantee the logical and temporal validity given and transactions to them. In that work, it introduces one mechanism of concurrency control for real-time database system, called is Semantic Concurrency Control Protocol, the concurrency control should have the ability to express the trade-off that results from the inherent conflict between temporal and logical consistency constraints, and also be able to maintain and bound any imprecision that results from trading off logical consistency for temporal consistency. This protocol is based on a Compatibility Function (CF), defined for the user, for the methods of reading and writing of an object of the application. The FC also controls and limits bounded imprecision when it happens trading off logical consistency for temporal consistency. The Semantic Concurrency Control Protocol was implemented in the programming language Java and the database was stored in the Database Management System of Microsoft DB2 IBM.

Índice

Glossário

BD	Banco de Dados
FC	Função de Compatibilidade
QoD	Qualidade dos Dados
QoS	Qualidade de Serviço
SGBD	Sistemas de Gerenciamento de Bancos de Dados Convencionais
SGBD-TR	Sistema de Gerenciamento de de Banco de Dados em Tempo-Real
STR	Sistemas em Tempo-Real
2TLP	Técnica de Bloqueio de Duas Fases Pessimista
CCO	Técnica de Controle de Concorrência Otimista
2TLP-HP	Técnica de Controle de Concorrência Pessimista - Alta Prioridade
2TLP-PE	Técnica de Controle de Concorrência Pessimista - Promovido à Espera
2TLP-HC	Técnica de Controle de Concorrência Pessimista - Herança Condicional
CCO-TC	Técnica de Controle de Concorrência Otimista - Transmissão Comprometida
CCO-BS	Técnica de Controle de Concorrência Otimista - Baseado em Sacrifício
CCO-BE	Técnica de Controle de Concorrência Otimista - Baseado em Espera
ES	Serialização *Epsilon*
AVI	ValidadeAbsoluta do Dado
IMP	Imprecisão Permitida

Lista de Figuras

Capítulo 1

Introdução

O avanço tecnológico das últimas décadas colocou de forma definitiva os computadores no cotidiano da vida moderna, seja num sistema bancário, de transporte, de saúde, de comunicação, de manufatura, entre tantos outros. Se de um lado tal inserção traz inúmeros benefícios às nossas vidas, de outro coloca novos desafios tecnológicos para a construção de sistemas computacionais cada vez mais confiáveis e seguros, sem os quais prejuízos e desastres tornar-se-ão inevitáveis (MACEDO et al., 2004).

Os Sistemas em Tempo-Real (STR) é um caso especial de sistemas computacionais em que o computador interage diretamente com o ambiente, não somente por meio de uma interface homem-máquina (vídeo e teclado), mas sobretudo através de dispositivos que captam informações e que interferem no ambiente, denominados de sensores. Exemplos desses sistemas vão desde simples fornos de microondas até sistemas de controle automático de vôo. Esses sistemas executam as ações desejadas dentro de intervalos de tempos determinados pelo ambiente. Por exemplo, o controle de pouso de um avião tem que acionar o trem de pouso tão logo seja detectado a proximidade do solo através de um sensor de altitude, caso contrário, pode resultar em conseqüências catastróficas.

Os Sistemas de Gerenciamento de Bancos de Dados Convencionais (SGBD) são projetados para gerenciar grandes volumes de dados e controlar a execução concorrente das transações, no entanto não suportam o tratamento das restrições de tempo inerentes em rede de sensores e em diversas aplicações tais como: sistemas de monitoramento de pacientes, sistemas de controle de trafego aéreo e ferroviário, bolsa de valores *on-line*, dentre outros (STANKOVIC; SON; HANSSON, 1999; NETO et al., 2004; LAM et al., 2002; LINDSTROM, 2002). Desta forma, pode ser inviável utilizar a tecnologia de SGBD convencional em tais aplicações.

Por outro lado, um Sistemas de Gerenciamento de Banco de Dados em Tempo-Real (SGBD-TR) possui como principais características para o tratamento de dados e transações com restrições explícitas de tempo. Tal característica impõe as técnicas de controle

de concorrência desses sistemas à necessidade de considerar tanto a manutenção da consistência lógica quanto a consistência temporal dos dados e transações (FERNANDES et al., 2004; RIBEIRO NETO et al., 2004). Entretanto, em algumas situações a manutenção simultânea de tais restrições é impraticável. Por exemplo, para manter a consistência temporal dos dados, uma transação que atualiza um item de dados pode interromper outra transação que esta lendo o mesmo item de dados. Se o item de dados em questão perdeu sua validade temporal, é interessante permitir que a sua consistência temporal seja restabelecida através da execução da transação de atualização. No entanto, esta execução poderia violar a consistência lógica do dado ou da transação de leitura. Então, existe a necessidade de se negociar entre manter consistência temporal ou consistência lógica. Se a consistência lógica for escolhida, então existe a possibilidade que o item de dados torne-se temporalmente inválido, ou que uma transação viole sua restrição temporal. Se por outro lado, consistência temporal for escolhida, a consistência lógica do dado ou da transação pode ser comprometida. Sendo necessário de uma técnica que permita a negociação entre a consistência lógica e consistência temporal dos dados e transações.

Em (DIPIPPO, 1995) foi definido uma técnica baseada em informação semântica, denominado *técnica de controle de concorrência semântico* que permite a negociação entre a consistência lógica e temporal dos dados e transações, e limita a imprecisão resultante através de um mecanismo denominado Função de Compatibilidade (FC) (DIPIPPO, 1995).

Neste trabalho, a técnica de controle de concorrência semântico foi modificada visando implementá-lo para rede de sensores. A linguagem de programação utilizada para a implementação foi Java (DEITEL; DEITEL, 2002), devido ser orientada a objetos, disponibilizar classes e interfaces necessárias para nossa aplicação. Após a implementação, foi realizado um estudo de caso focado em uma aplicação de rede de sensores, onde analisamos as restrições lógicas e temporais dos dados e das transações resultantes da execução concorrente das transações.

1.1 Motivação

A simples integração dos SGBD e STR, considerando conceitos, mecanismos e ferramentas, não é suficiente para desenvolver um SGBD-TR. Apesar de muitas das técnicas usadas em STR e SGBD poderem ser aplicadas aos SGBD-TR, há muitas diferenças que requerem adaptações das abordagens usadas nas duas áreas, ou mesmo, o desenvolvimento de novas abordagens. Questões como modelagem conceitual, controle de concorrência, escalonamento, recuperação, linguagem de consultas, entre outras, estão sendo consideradas e pesquisadas para SGBD-TR (PERKUSICH, 2000; KANG, 2001; LINDSTROM, 2003; RIBEIRO NETO, 2005; LEITE et al., 2005).

A necessidade de disponibilizar uma técnica de controle de concorrência para tratar com aplicações em tempo-real, motivou a implementar uma técnica de controle de concorrência para os SGBD-TR que pode substituir a técnica de controle de concorrência dos SGBD tradicionais.

A linguagem de programação Java (DEITEL; DEITEL, 2002), foi escolhida devido a ser orientada a objetos, independente de plataformas, suporta acesso à banco de dados (JDBC) (DEITEL; DEITEL, 2000), além de permitir vários recursos de alto desempenho (multithreading) (LEA, 1999) e extensões para tempo-real (Real-Time Java) (DIBBLE, 2002).

1.2 Declaração do Problema

Nos últimos anos, várias pesquisas têm sido voltadas para técnicas de controle de concorrência para SGBD-TR (SON; LEE, 1994; CHIU; KAO; LAM, 1998; LAM et al., 2002; LINDSTROM, 2003). Muitas dessas técnicas são baseadas em técnicas de controle de concorrência tradicionais. A maioria dessas técnicas baseia-se no mecanismo de *serialização* como critério de corretude para determinar quais as transações que podem executar concorrentemente (SUDARSHAN; KORTH; SILBERSCHATZ, 2001).

As técnicas de controle de concorrência mais utilizadas para SGBD-TR são:

- Técnica de Controle de Concorrência Baseada em Bloqueio de Duas Fases Pessimista (2TLP) (NYSTRöM et al., 2004);
- Técnica de Controle de Concorrência Otimista (CCO) (LINDSTROM, 2002, 2003; RAATIKAINEN; LINDSTROM, 2002).

Na Técnica de Bloqueio de Duas Fases Pessimista as transações adquirem os bloqueios antes de executarem suas operações no banco de dados ou esperam pelo bloqueio se este não puder ser adquirido. No entanto, essa técnica pode gerar tempos indeterminados de espera pelo bloqueio. Dessa forma, a utilização desse mecanismo pode ser vantajosa por manter a consistência lógica do banco de dados sem impasses, em contrapartida pode comprometer as restrições de tempo impostas às transações.

A outra técnica utilizada é a Técnica de Controle de Concorrência Otimista, onde a resolução de conflito é atrasada até a transação está próxima de comprometer. Nessa técnica não existe a indeterminação do tempo de espera, porém a quantidade de transações que podem ser reiniciadas é grande, o que pode ser fatal em sistemas com restrições temporais.

Em SGBD-TR, os aspectos temporais dos dados e as restrições de tempo das transações também devem ser considerados. Portanto, uma técnica de controle de concorrência

para esses sistemas deve também manter a *consistência temporal dos dados e transações* (DIPIPPO, 1995). De forma que, suporte a consistência lógica e temporal dos dados e transações, além de permitir a negociação entre elas, possibilitando expressar a imprecisão resultante dessa negociação.

Inicialmente foram realizadas algumas pesquisas apontando que a técnica de controle de concorrência semântica é adequado para os SGBD-TR (PERKUSICH, 2000). Com já mencionadao, essa técnica é baseada em uma FC, que faz a negociação entre a manutenção da consistência lógica ou consistência temporal, e controlar a quantidade de imprecisão resultante dessa negociação. Em trabalhos recentes, também foram feitas pesquisas com base nessa técnica e obtiveram bons resultados, entre eles:

- Em (RIBEIRO NETO, 2001), foi feita uma análise formal de um modelo para SGBD-TR que utiliza essa técnica em uma aplicação buscando determinar quais transações podem ser executadas concorrentemente.

- Em (RIBEIRO NETO; PERKUSICH; PERKUSICH, 2003b; RIBEIRO NETO; PERKUSICH; M.L.B., 2003), foram feitas análises com mecanismos de qualidade de serviços (QoS) para os SGBD-TR, e obtiveram alguns resultados para análise. Foi verificado um aumento considerável de execuções concorrentes, mantendo a consistência do SGBD-TR.

- Em (RIBEIRO NETO, 2003; RIBEIRO NETO; PERKUSICH; PERKUSICH, 2003a) foi definido formalmente a função de compatibilidade, usada para expressar quais transações podem ser executadas concorrentemente, considerando propriedades de QoS a fim de garantir a corretude do sistema mesmo em situações onde os limites de imprecisão foram ultrapassados.

1.3 Objetivos da Dissertação

O objetivo principal desse trabalho é redefinir e implementar um algoritmo de uma técnica de controle de concorrência semântico para controlar a execução concorrente das transações de um SGBD-TR. Tal técnica suporta a consistência temporal das transações por permitir a especificação de escalonamentos mais flexíveis que aqueles permitidos por serialização e suas variações (RAMAMRITHAM; PU, 1995). Como citado na Seção 1 esta técnica é baseada no algoritmo apresentado em (DIPIPPO, 1995) que tem como base uma FC que controla a execução concorrente de transações conflitantes e limita a imprecisão resultante quando as mesmas são executadas. Visando validar a corretude da técnica implementada, foi realizado um estudo de caso focando em rede de sensores.

1.4 Relevância

A relevância desse trabalho é disponibilizar classes em Java da técnica de controle de concorrência semântico, que pode ser utilizado em aplicações que necessitam tratar com dados e transações com restrições temporais e que precisam negociar entre a consistência lógica e consistência temporal. É importante observar, que tais aplicações podem ser controladas por qualquer gerenciador de banco de dados, entretanto a técnica de controle de concorrência desses gerenciadores precisam ser desativadas para permitir que a técnica implementada seja utilizada para gerenciar o controle de concorrência das transações.

1.5 Organização da Dissertação

Capítulo 2

Nesse capítulo apresenta-se os conceitos básicos de Sistemas em Tempo-Real (STR), Sistemas de Gerenciamento de Banco de Dados Convencionais (SGBD) e Sistemas de Gerenciamento de Banco de Dados em Tempo-Real (SGBD-TR), nas Seções 2.1, 2.2 e 2.3,respectivamente. Considerando que o objetivo principal deste trabalho é a implementação de uma técnica de controle de concorrência semântica para SGBD-TR, foi dado uma maior ênfase aos mesmos.

Capítulo 3

Nesse capítulo, é apresentado as Técnicas de Controle de Concorrência para SGBD-TR. Inicialmente, na Seção 3.1, o Problema da Execução Concorrente das Transações são apresentados. Na Seção 3.2, são apresentados os conceitos básicos das Técnicas de Controle de Concorrência para SGBD tradicionais. Na Seção 3.4, é apresentado as Técnicas de Controle de Concorrência para SGBD-TR. Na Seção 3.6, é apresentada um comparativo entre Técnica de Controle de Concorrência Otimista e Técnica de Controle de Concorrência Pessimista para SGBD-TR.

Capítulo 4

Nesse Capítulo é apresentada uma Técnica de Controle de Concorrência Semântico para SGBD-TR. Inicialmente, na Seção 4.1 é apresentado a motivação de utilizar essa Técnica. Em seguida 4.2, é apresentada a Técnica de Controle de Concorrência Semântico para SGBD-TR, sendo o foco do nosso trabalho. Nas Seções 4.3, 4.4 é apresentado os mecanismos em que a Técnica de Controle de Concorrência é baseada que é Serialização *Epsilon* e Função de Compatibilidade, respectivamente. Na Seção 4.5 é apresentado o algoritmo

da Técnica de Controle de Concorrência, e finalmente na Seção 4.5.3 a redefinição do algoritmo.

Capítulo 5

Nesse Capítulo, é apresentado a implementação da Técnica de Controle de Concorrência enfatizando a Função de Compatibilidade que é base do mesmo. Também é apresentado um estudo de caso focado em Rede de sensores visando validar o método implementado. Inicialmente, na Seção 5.1, é apresentado a descrição da implementação da técnica e, em seguida é apresentado o estudo de caso na Seção 5.2. Finalmente na Seção 5.3 é apresentado a avaliação do desempenho da técnica implementada.

Capítulo 6

Nesse Capítulo apresenta-se as conclusões e as propostas de trabalhos futuros.

Capítulo 2

Sistemas de Gerenciamento de Banco de Dados em Tempo-Real

Nesse capítulo apresenta-se os conceitos básicos de Sistemas em Tempo-Real (STR), Sistemas de Gerenciamento de Banco de Dados Convencionais (SGBD) e Sistemas de Gerenciamento de Banco de Dados em Tempo-Real (SGBD-TR), nas Seções 2.1, 2.2 e 2.3,respectivamente. Considerando que o objetivo principal deste trabalho é a implementação de uma técnica de controle de concorrência semântico para SGBD-TR, foi dado uma maior ênfase aos mesmos.

2.1 Introdução

2.1.1 Sistemas em Tempo-Real

Os STR são sistemas que devem reagir a estímulos oriundos de um ambiente dentro de prazos específicos (FARINES; FRAGA; OLIVEIRA, 2000). Assim, para que uma tarefa em um STR seja executada com sucesso, ela deve ser executada dentro do intervalo de tempo determinado pelo sistema. Para isso, as tarefas de um STR devem incluir declarações explícitas sobre o tempo no qual o processamento deve ser realizado (PERKUSICH, 2000). Um STR consiste de um sistema controlador e um sistema controlado (LINDSTROM, 2003). O sistema controlado é o ambiente com o qual o computador e o software interagem. O sistema controlador interage com o ambiente através dos dados adquiridos do mesmo como está ilustrado na Figura 2.1. O sistema controlador interage com o sistema controlado com base nos dados disponíveis sobre o ambiente, os quais são obtidos de vários sensores, por exemplo, sensor de temperatura. É extremamente importante que o sistema controlador obtenha informações sobre o estado real do ambiente de forma que os resultados obtidos sejam os esperados, de outra forma, estes podem ser catastróficos. Conseqüentemente, o

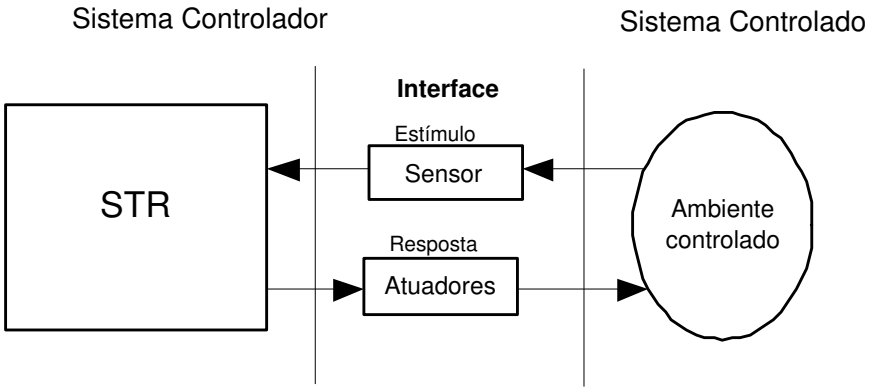

Figura 2.1: Sistema Controlador e Sistema Controlado

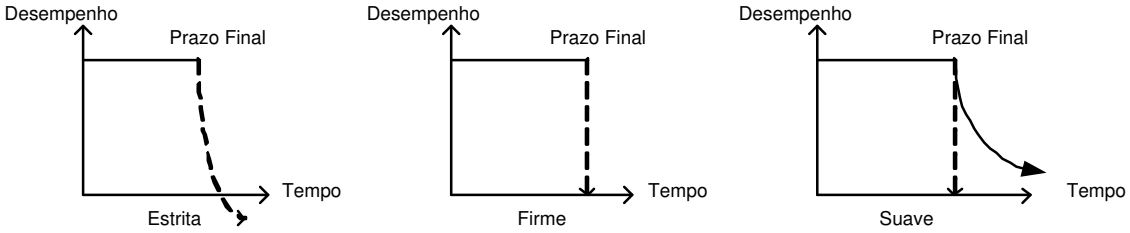

Figura 2.2: Tipos de Prazos Finais

monitoramento em tempo real do ambiente bem como o processamento em tempo real das informações adquiridas do ambiente é necessário. Assim, a corretude de um STR não depende apenas dos resultados lógicos, mas também do tempo no qual os resultados são produzidos (AL-OMARI; SOMANI; MANIMARAN, 2004).

Aplicações em tempo-real são caracterizadas por restrições de tempo que devem ser respeitadas para obterem o comportamento temporal desejado ou necessário. Tipicamente as tarefas de um STR estão sujeitas a prazos[1] (LIU, 2000). Um STR pode ser caracterizado pelo seus tipos de restrições de tempo, como pode ser observada na Figura 2.2:

- Na Figura 2.2(A) é ilustrada uma tarefa com prazo estrito: uma tarefa tem prazo estrito quando qualquer resultado produzido após seu prazo final é inútil para o sistema. Isso significa que qualquer tarefa estrita deve ser abortada quando não puder cumprir seu prazo, independentemente de sua conseqüência. Quando uma tarefa com prazo estrito perde seu prazo gera um valor negativo para os sistema e pode resultar em algo catastrófico.

- Na Figura 2.2 (B) é ilustrada uma tarefa com prazo firme: uma tarefa com prazo firme mesmo perdendo seu prazo final não gera nenhum efeito ou valor negativo para o sistema. Geralmente estas tarefas são reiniciadas quando perdem seu prazo,

[1] *Deadline.*

ou seja, não degrada o desempenho do sistema.

- Na Figura 2.2 (C) é ilustrada uma tarefa com prazo suave: uma tarefa tem prazo suave quando o resultado produzido após seu prazo final sempre tem sempre algum valor, que vai perdendo sua utilidade a medida que se distancia do seu prazo final.

Existe outras restrições temporais que são importantes na definição do comportamento temporal de uma tarefa:

- Tempo Computacional: o tempo computacional de uma tarefa é o tempo necessário para a execução completa da tarefa.
- Tempo de Início: corresponde ao instante de inicio do processamento da tarefa em uma ativação.
- Tempo de Término: é o instante de tempo em que se completa a execução da tarefa.
- Tempo de Chegada: o tempo de chegada de uma tarefa é o instante em que o escalonador toma conhecimento dessa tarefa.
- Tempo de Liberação: o tempo de liberação de uma tarefa coincide com o instante de sua inclusão na fila de tarefas prontas para executar.

As aplicações em tempo-real com prazos estritos necessitam conhecer a priori o comportamento do sistema, de forma que seja possível prever se os prazos de suas tarefas serão atendidos. Dessa forma, a previsibilidade é um requisito necessário que dever ser introduzido para atender tais tipos de aplicações.

2.1.2 Previsibilidade nos Sistemas em Tempo-Real

Pode-se definir um sistema como previsível se o seu comportamento pode ser de algum modo antecipado ou previsto (FARINES; FRAGA; OLIVEIRA, 2000). Para que o sistema funcione de forma previsível é necessário, em primeiro lugar, que haja um gerenciamento adequado dos recursos disponíveis a fim de que as ações em tempo-real não sofram atrasos imprevistos, por exemplo, devido à indisponibilidade de memória, CPU ou mesmo a execução simultânea de diferentes tarefas. Os sistemas que apresentam restrições temporais estritas, como controles de plantas nucleares e controle de vôo, requerem uma previsibilidade próxima de cem por cento. Por outro lado, a previsibilidade em sistemas convencionais (fora do escopo de tempo-real), como, por exemplo, consultas num cadastro de uma empresa, podem ser realizadas num intervalo de tempo muito menos rígido, porém o melhor possível.

Nos últimos anos novas aplicações adotaram sistemas onde a sobrecarga[2] computacio-

[2] *Overload.*

nal não pode ser conhecida a priori, ou seja, não conseguem garantir que todas as tarefas serão executadas dentro dos prazos especificados. Tais sistemas são denominados sistemas abertos e imprevisíveis.

2.1.3 Imprecisão em Sistemas em Tempo-Real

Em sistemas onde o tempo de execução de uma tarefa dentro do prazo definido é mais importante que o resultado exato fora do prazo, um resultado aproximado pode ser aceitável, ou seja, a imprecisão pode ser tolerada. Por exemplo, em um sistema de controle de tráfego aéreo, a posição aproximada de um avião no instante certo, pode ser mais apropriada que o valor correto tarde demais. Geralmente a imprecisão permitida deve ser limitada. Considerando o exemplo anterior, a posição aproximada do avião deve ser diferente da verdadeira posição em apenas alguns metros. É importante observar que os STR consideram as restrições temporais das tarefas mas ignoram problemas de consistência lógica dos dados.

2.2 Sistemas de Gerenciamento de Banco de Dados

Um Banco de Dados (BD) é uma coleção de dados inter-relacionados relativos a um domínio específico. Um sistema de gerenciamento de banco de dados (SGBD) é uma coleção de programas usados para acessar o banco de dados (LINDSTROM, 2003). Os SGBD são caracterizados por admitir acesso concorrente e eficiente a grandes volumes de dados enquanto mantém a corretude dos mesmos, suportar linguagens de consultas declarativas para definir as operações que devem acessar o BD, e permitir que os dados persistam no BD com segurança. Um SGBD tradicional apresenta os seguintes componentes (HOLANDA; BRAYNER; FIALHO, 2004), como mostrado na Figura 2.3.

- O gerenciador de cache: tem como finalidade manter o cachê, movendo dados da unidade de armazenamento volátil para a estável, em resposta às solicitações dos níveis mais altos;
- O gerenciador de recuperação: é responsável por garantir que o banco de dados mantenha todos os efeitos de uma transação confirmada e nenhum dos efeitos de uma transação abortada;
- Técnica de controle de concorrência: tem por objetivo de controlar a execução concorrente das transações;
- Gerenciador de transações: tem como função básica, receber as transações e encaminhá-las para a técnica de controle de concorrência.

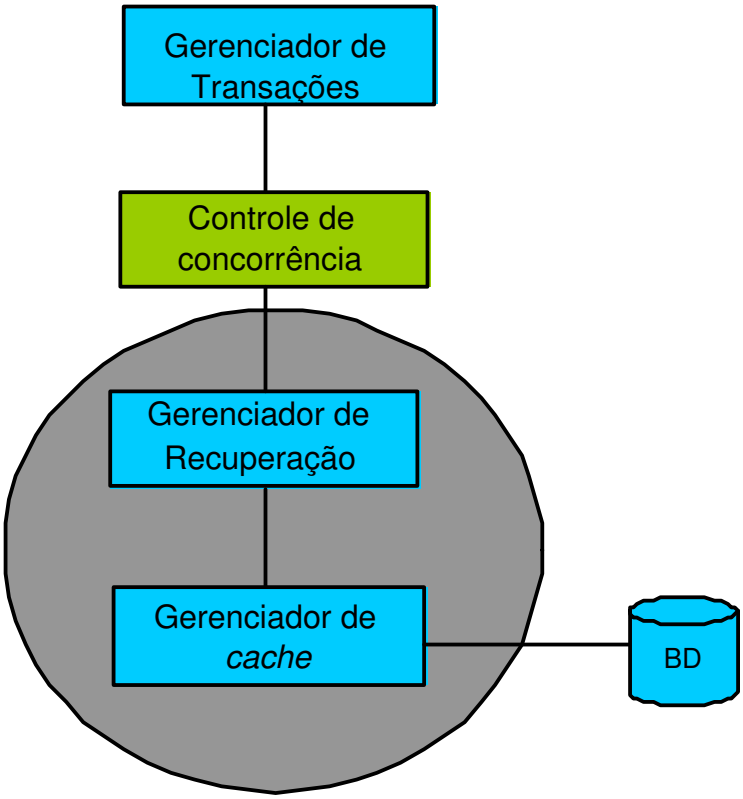

Figura 2.3: SGBD Tradicional

Considerando que o objetivo principal deste trabalho é no componente da técnica de controle de concorrência.

Os BD consistem de uma coleção de objetos que representam entidades do mundo real. Cada objeto do BD tem um nome e um valor. Por exemplo, um objeto chamado de conta A com o valor de 5.000 pode representar um conta de um determinado banco. O conjunto de valores de todos os objetos armazenados em um BD num determinado instante é chamado de estado do BD.

Assim, o estado de um BD representa um visão instantânea do mundo, refletindo apenas seus aspectos estáticos. Entretanto, as mudanças que ocorrem no mundo real devem ser refletidas no BD. Essas mudanças são capturadas pelo conceito de transição de estados. Uma transição de estado representa o salto de um determinado estado para outro, estas são realizadas pelos programas aplicativos que contem operações de leitura e escrita sobre os objetos do banco de dados (FILHO, 2001).

Transações são conjuntos de operações que acessam ou modificam o conteúdo do BD, sendo usadas para representar operações sobre o BD executadas pela aplicações. Afim de preservar a semântica dos programas aplicativos, as transações também indicam a ordem na qual as operações sobre um BD devem se executadas. Portanto, quando uma transação é submetida ao SGBD, este deve garantir que todas as operações desta transação serão

completadas com sucesso e seus resultados gravados com *persistência* no BD. Portanto, o SGBD precisa acompanhar o momento no qual uma transação começa e termina, visando identificar quando ela termina com êxito ou precisa ser abortada. Uma transação tem diferentes estados: (i) Ativa - uma transação em execução; (ii)Abortada - uma transação que inicia mas não é confirmada deve ser abortada;(iii) Reiniciada - quando uma transação é abortada, ela pode ser reiniciada.

Uns dos principais objetivos dos SGBD é permitir que vários transações acessem dados compartilhados concorrentemente (LINDSTROM, 2003). Entretanto, quando várias transações estão acessando concorrentemente o SGBD, os resultados produzidos ou recuperados pelas mesmas podem ser incorretos. Como conseqüência, a integridade e a consistência do BD pode ser violada (HUNG; HUONG, 2002).

Para garantir a integridade dos banco de dados foram definidas propriedades, comumente chamadas de propriedades ACID[3] (ELMASRI; NAVATHE, 2002).

- Atomicidade: uma transação é uma unidade atômica de processamento, ou ela é realizada em sua totalidade ou não é realizada de modo algum.
- Consistência: a execução de uma transação deve deixar o BD de um outro estado consistente - resultante também da propriedade isolamento.
- Isolamento: uma transação não deve deixar suas atualizações visíveis para outras transações até que seja comprometida.
- Durabilidade: quando uma transação modifica o BD e sua modificações são comprometidas, essas modificações não podem ser perdidas.

A propriedade de isolamento de uma transação garante que os resultados parciais produzidos por uma transação não serão visíveis as transações que estão sendo executadas concorrentemente até que ela seja comprometida, ou seja, elas comportam-se como se estivessem sendo executadas uma de cada vez. Assegurar a propriedade de isolamento é responsabilidade de um componente do SGBD denominado de técnica de controle de concorrência, que será apresentado com mais detalhes no capítulo 3.

Portanto, o objetivo fundamental de um SGBD é maximizar o grau de concorrência no sistema (LINDSTROM, 2003), garantindo que qualquer escalonamento de transações executado concorrentemente tenha o mesmo efeito de um escalonamento serial, ou seja, equivalente a execução de uma transação após a outra.

Assim, *a teoria da serialização* determina quais escalonamentos concorrentes são corretos e quais não são, e procura desenvolver técnicas que permitam executar apenas es-

[3]É o acrônimo para *Atomicidade, Consistência, Isolamento e Durabilidade.*

Transação T1	Transação T2	Há conflito?
Read (X)	Read (X)	Não
Read (X)	Write (X)	Sim
Write (X)	Read (X)	Sim
Write (X)	Write (X)	Sim

Figura 2.4: Conflitos entre Operaçãoes das Transações

calonamentos serializáveis, que produzam resultados corretos (SUDARSHAN; KORTH; SILBERSCHATZ, 2001).

2.2.1 Serialização

Existe conflitos entre duas operações de uma transação se satisfazem três condições: (1) pertencer a diferentes transações; (2) acessar o mesmo item de dado; (3) pelos menos uma das operações da transação deve ser de escrita. Para um melhor entendimento, representamos as transações por uma notação abreviada, onde: seja as transações $r_{T1}(X)$ e $w_{T1}(X)$, indicando que a operação da transação $T1$ consulta ou atualiza respectivamente, o item de dados X. Para a transação de $T2$, adotamos $w_{T2}(X)$ e $r_{T2}(X)$, indicando que a operação da transação consulta ou atualiza o item de dado$X$. Como ilustrado na seguinte Figura 2.4.

1. Quando a operação de $r_{T1}(X)$ consulta um item de dado X e a operação da transação de $T2$, $w_{T2}(X)$ atualiza o item de dado X.

2. Quando a operação da transação de $T1$, $w_{T1}(X)$ atualiza o item de dado X e a operação da transação de $T2$,$r_{T2}(X)$ faz consulta no mesmo item de dado.

3. Quando as operações das transações $w_{T1}(X)$ e $w_{T2}(X)$ querem atualizar o mesmo item de dado X.

No entanto, as operações $r_{T1}(X)$ e $r_{T2}(X)$ não têm conflitos, uma vez que ambas são operações de consulta; as operações $w_{T1}(X)$ e $w_{T2}(Y)$ não têm conflitos, porque operam em diferentes ítens de dados X e Y; e as operações $r_T(X)$ e $w_T(X)$ não têm conflitos porque pertencem a mesma transação.

Se a intercalação entre a execução das operações das transações não for possível, haverá somente duas maneiras de ordenar as operações das transações:

1. Executar todas as operações de T1 (em seqüência) seguidas por todas as operações de T2 (em seqüência);

2. Executar todas as operações de T2 (em seqüência) seguidas por todas as operações de T1 (em seqüência);

Se a intercalação das operações for permitida, então existirão muitos escalonamentos possíveis no qual o sistema poderá executar as operações das transações.

Um escalonamento é serial se, para cada transação T participante do escalonamento, todas as operações de T são executadas consecutivamente no escalonamento; caso contrário, o escalonamento é não serial (ELMASRI; NAVATHE, 2002). O problema com escalonamento serial é a sua limitação quanto a intercalação das operações. No escalonamento serial, se uma transação espera por uma operação de entrada e saída para terminar, não se pode mudar o processador para executar outra transação, isso desperdiça um valioso tempo de processamento e faz com que o escalonamento serial não seja aceitável em um ambiente de banco de dados. É interessante determinar quais escalonamentos não-seriais dão o resultado correto e quais dão resultados errados. O conceito usado para caracterizar escalonamentos dessa maneira é o de *serialização*(ELMASRI; NAVATHE, 2002).

Um escalonamento de transações é *serializável* se for equivalente a um escalonamento serial. Pode-se então formar dois grupos diferentes de escalonamentos não-seriais: os que são equivalentes a um (ou mais) escalonamento serial, sendo*serializável*; e os que não são equivalentes a qualquer escalonamento serial, não sendo *serializável*.

Dessa forma, as técnicas de controle de concorrência tem por objetivo proporcionar um alto grau de concorrência garantindo que todos os escalonamentos são *serializáveis*. Para manter a *serialização*, algumas transações podem ser atrasadas e, conseqüentemente perderem seus prazos. Porém, em se tratando de aplicações que envolvam restrições de tempo que é o foco do nosso trabalho, é preferível um valor aproximado em um determinado instante de tempo a um valor exato sem validade temporal. Por exemplo, uma transação pode precisar da média aproximada dos saldos das contas correntes de um banco financeiro, durante o expediente bancário, para realização de um negócio. Para atender a esses propósitos, o padrão SQL, usado pelos banco de dados convencionais, já permite o relaxamento da *serialização* em vários níveis, embora que não considere restrições de tempo.

2.2.2 Níveis de Isolamento

A fim de manter um maior grau de concorrência em SGBD, o padrão SQL-99 permite a especificação de diferentes níveis de isolamento para o tratamento da execução concorrente de transações. Os níveis permitidos são (SUDARSHAN; KORTH; SILBERSCHATZ, 2001):

- Leitura sem efetivação (*Read Uncommitted*): permite a leitura de registros que não sofreram efetivação. É o nível mais fraco de isolamento permitido pelo padrão SQL;
- Leitura com efetivação (*Read Committed*): permite que apenas dados que sofreram efetivação sejam lidos. Por exemplo, entre duas leituras de dados feitas por uma transação, os dados podem ter sido atualizados por meio de outras transações que obtiveram efetivação;
- Leitura repetitiva (*Repeatable Read*): somente permite a leitura de dados que sofreram efetivação e, além disso, exige que nenhuma outra transação consiga atualizar um dado entre duas leituras feitas por uma transação;
- Serialização (*Serializable*): este é o nível de segurança máximo no isolamento de transações. Garante que todas as transações são executadas em série, isto é, possuem acesso de leitura e escrita exclusivo para um registro até que salvem os resultados outras transações precisam aguardar. Pode diminuir muito o desempenho do sistema eliminando a execução concorrente.

Desta forma, o nível definido para uma aplicação deve ser escolhido de forma bastante cautelosa. Por exemplo, o nível de isolamento *serializável* garante a corretude dos dados, mais diminui a concorrência. Por outro lado, o nível de isolamento *leitura sem efetivação* não garante a consistência dos dados, no entanto gera um alto grau de concorrência.

Apesar da SQL-99 permitir a definição de diferentes níveis de isolamento, nenhum controle é realizado em relação à quantidade de inconsistência lógica adicionada no banco de dados e nenhum aspecto de consistência temporal é considerado.

2.3 Sistemas de Gerenciamento de Banco de Dados em Tempo-Real

Nos últimos anos, várias pesquisas têm sido realizadas na área de SGBD-TR (PERKUSICH, 2000; FERNANDES et al., 2004; NETO et al., 2004; RIBEIRO NETO et al., 2004; KANG; SON; STANKOVIC, 2004), uma vez que esses sistemas tratam com aplicações que envolvem o gerenciamento de grandes quantidades de dados, além de permitir tratar com dados e transações com restrições em tempo-real. Os SGBD-TR surgiram com uma publicação

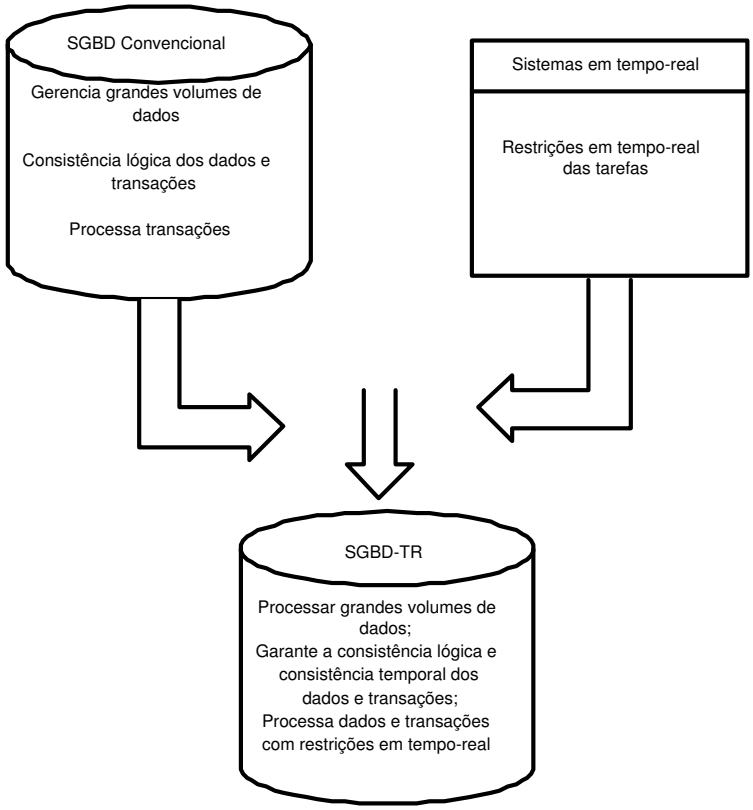

Figura 2.5: Características de um SGBD-TR

especial, registrado em *ACM SIGMOD* em março 1988. Atualmente, muitas aplicações requerem o uso de tais sistemas, dentre elas: os sistemas de recuperação da informação, os sistemas do reserva de passagem aérea, sistemas bancários, sistemas de controle de avião e de nave espacial, robótica, a automação de fábrica, dentre outros (SQUADRITO, 1996). No entanto, existe outra área de pesquisa onde o tempo é considerado são os Banco de Dados Temporais (JENSEN, 2000).

A importância funcional dos SGBD-TR se dá pelo fato destes suportarem as características de SGBD, de processar transações e garantir a consistência lógica dos dados e transações, como também as características de um STR, de satisfazer às restrições de tempo impostas as tarefas, com ilustrado na Figura 2.5. Assim, como os SGBD tradicionais, os SGBD-TR servem como repositórios para dados e permitem o processamento eficientes dos mesmos.

As principais diferenças entre um SGBD tradicional e um SGBD-TR são: o gerenciamento de dados temporalmente consistentes, ou seja, dados que são válidos apenas por um período de tempo específico e o gerenciamento de transações de forma que elas executem dentro de um prazo definido a priori. As restrições temporais podem ser originadas da necessidade de rastrear continuamente um ambiente ou da necessidade de fazer com que os dados do sistema controlado estejam disponíveis para suas atividades de tomada de

decisão (GRAHAM, 1993).

2.3.1 Tipos de Dados

A consistência temporal dos dados exige que o estado atual do ambiente da aplicação e o estado representado pelo conteúdo do banco de dados devam ser próximos o bastante para permanecer em um limite de tolerância aceitável pelas aplicações. A consistência temporal tem dois componentes: consistência absoluta e consistência relativa (BESTRAVOS; LIN; S.H., 1997).

A *consistência absoluta* é medida entre o estado do ambiente e como ele é refletido no banco de dados. Então, um item de dado deve ser gravado dentro de intervalos de tempo que garantam que ele reflete o estado real do ambiente. Essa medida surge da necessidade de manter a visão do sistema consistente com o estado real do ambiente. Por exemplo, em um sistema de transporte automatizado, o dado correspondente a um sensor que verifica a velocidade dos veículos deve ser atualizado periodicamente, por exemplo, a cada cinco minutos. Assim, o valor da velocidade é absolutamente consistente se ele foi gravado nos últimos cinco minutos.

Já a *consistência relativa* é medida entre os dados que são usados na computação de outros dados. Então, um conjunto de itens de dados deve ser gravado dentro de um mesmo intervalo de tempo que possa representar aproximadamente o mesmo instante de tempo. Essa medida surge da necessidade de produzir novos dados a partir de dados gravados em tempos aproximados. Por exemplo, se o sistema de transporte computa o valor dos níveis de consumo de combustível usando os valores da *velocidade* e da *posição atual* em que se encontra um avião, é importante que esses valores tenham sido gravados aproximadamente no mesmo tempo, por exemplo, nos últimos trinta segundos, de forma que eles possam refletir o mesmo instante do ambiente, ou a computação não fará sentido.

2.3.2 Tipos de Transações

Um SGBD-TR deve gerenciar não apenas a consistência lógica dos dados e transações, mas também deve satisfazer as restrições de tempo das transações. Tais restrições incluem *prazo final, tempo de inicio mais cedo* [4] *e tempo de inicio mais tarde* [5]. As transações devem ser escalonadas de acordo com suas restrições.

Existem várias maneiras de classificar as transações em tempo-real. Elas podem ser classificadas quanto as restrições de tempo-real, ou seja, no efeito de perder seu prazo em: estrita, suave e firme, como para STR.

[4] Earliest Start Time
[5] Latest Start Time

Com relação ao intervalo de tempo em que as transações podem executar, elas podem ser: periódicas, aperiódicas e esporádicas. Geralmente, em SGBD-TR as transações são executadas periodicamente, ou seja, existe um intervalo de tempo regular entre as execuções das transações. As transações também podem ser aperiódicas, onde não existe um intervalo de tempo regular entre suas execuções. Por fim, as transações esporádicas têm um período para iniciar, podendo ou não executar. (LIU, 2000).

Quanto ao tipo, as transações podem ser classificadas como: transações de escrita, transações de atualização e transações de leitura (RAMAMRITHMAN, 1993). As transações de escrita obtém o estado do ambiente e escrevem os dados no banco de dados. Essas são tipicamente periódicas. As transações de atualização tanto podem ler quanto escrever no banco de dados. As transações de leitura apenas lêem dados do banco de dados.

2.3.3 Processamento de Transações

Técnicas de processamento de transações que consideram as características temporais dos dados e das transações têm sido uma das principais pesquisas em SGBD-TR (RAMAMRITHMAN; SON; DIPIPPO, 2004). Muitas dessas pesquisas consideram a negociação inerente entre a consistência dos lógica e temporal dos dados e o tempo de processamento das transações.

Por exemplo, em uma aplicação de bolsas de valores, uma transação que atualiza um item de dado deve executar concorrentemente com outra transação que esta lendo o mesmo item de dado, e tem um prazo final pequeno. Se o item de dado da bolsa de valores em questão é "velho"é interessante permitir que a sua consistência temporal seja restabelecida através da execução da transação de atualização (RAMAMRITHMAN; SON; DIPIPPO, 2004). No entanto, esta execução poderia violar a consistência lógica do dado ou da transação de leitura. Então, existe a necessidade de se negociar entre manter consistência temporal ou consistência lógica. Se a consistência lógica for escolhida, o item de dado pode tornar-se temporalmente inválido, ou a transação pode perder o seu prazo final. Se por outro lado, se a consistência temporal for mais importante, a consistência lógica do dado é sacrificada visando manter a consistência temporal da transação.

Capítulo 3

Técnica de Controle de Concorrência para SGBD-TR

Neste capítulo, é apresentado Técnica de Controle de Concorrência para SGBD-TR. Inicialmente, na Seção 3.1, o Problema da Execução Concorrente de Transações são apresentados. Na Seção 3.2, são apresentados os conceitos básicos das Técnicas de Controle de Concorrência para SGBD convencionais. Na Seção 3.4, é apresentado as Técnicas de Controle de Concorrência para SGBD-TR. Na Seção 3.6, é apresentada um comparativo entre Técnica de Controle de Concorrência Otimista e Técnica de Controle de Concorrência Pessimista para SGBD-TR.

3.1 Introdução

Desde os anos setenta o problema de controle de concorrência em ambientes multiusuário tem sido amplamente explorado (FILHO, 2001). Em 1976, propuseram um modelo que introduziu o conceito de transação. Este modelo, conhecido como modelo clássico para o controle de concorrência, é baseado no princípio de que a execução de uma transação em ambiente multiusuário deve ser atômica, ou seja sem interferência ou entrelaçamento com operações de outras transações.

O modelo clássico de transações tem se consolidado com uma solução padrão para o problema do controle de concorrência em SGBD (ELMASRI; NAVATHE, 2002). Esse modelo tem mostrado apropriado para o processamento de transações em aplicações convencionais de BD nas quais as transações são de curta duração e o armazenamento dos dados é centralizado.

Contudo, a natureza e o requisito do processamento de transações em ambientes de tempo-real diferem das aplicações convencionais. Desta forma as técnicas de controle de concorrência convencionais são inadequadas para aplicações em tempo-real. Conseqüente-

mente as transações e os dados para esse tipo de aplicações devem ter restrições temporais que devem ser respeitadas.

3.1.1 O Problema da Execução Concorrente das Transações

As aplicações quando são executadas concorrentemente, isto é, através do entrelaçamento de operações de diferentes programas, pode levar a mudanças inconsistentes no banco de dados. Consequentemente, a execução concorrente de programas aplicativos deve ser controlada e monitorada. Está funcionalidade é chamada controle de concorrência.

A execução concorrente de um conjunto de transações é realizada através do entrelaçamento das operações que compõem as várias transações. No entanto, alguns entrelaçamentos podem produzir estados inconsistentes, sendo necessário definir quando a execução concorrentes de transações resulta em estados consistentes.

Assim, se uma execução concorrente de um conjunto de transações é equivalente à execução serial das mesmas transações, então ela também é correta. Esse critério de corretude é chamado serialização, como apresentado no Capitúlo 2.

Dessa forma, a serialização proporciona a ilusão que a execução concorrente de múltiplas transações acontece de forma serial, uma após a outra. Por está razão, dizemos que a serialização fornece a ilusão que a execução de uma transação consiste em uma ação atômica, ou seja, ou todas as operações de uma transação são executadas com sucesso ou nenhuma delas deverá ser executada, gerando o aborto da mesma.

Portanto, o principal objetivo do controle de concorrência é maximizar a concorrência entre as transações ao mesmo tempo em que garante a consistência do BD. Dessa forma, a execução entrelaçada de um conjunto de transações é controlada através de técnicas de controle de concorrência. As técnicas de controle de concorrência indica a ordem na qual as operações de um conjunto de transações são executadas umas com relação as outras, ou seja, representa uma execução concorrente.

3.2 Técnica de Controle de Concorrência Tradicional

Os sistemas de banco de dados tradicionais são projetados para permitir que várias transações acessem itens de dados no banco de dados de forma concorrente. Isto é feito por meio de uma variedade de mecanismos chamados de técnicas de controle de concorrência(SUDARSHAN; KORTH; SILBERSCHATZ, 2001).

Assim, o processamento de transações são implementados através das técnicas de controle de concorrência. Estas técnicas são responsáveis por produzir escalonamentos em conformidade com o critério de corretude adotado. Dessa forma, uma técnica de controle de concorrência recebe como entrada as operações pertencentes a um determinado

conjunto de transações e produz com saída uma seqüencia corretemente serializada destas operações (FILHO, 2001). Como ilustrado na Figura 3.1.

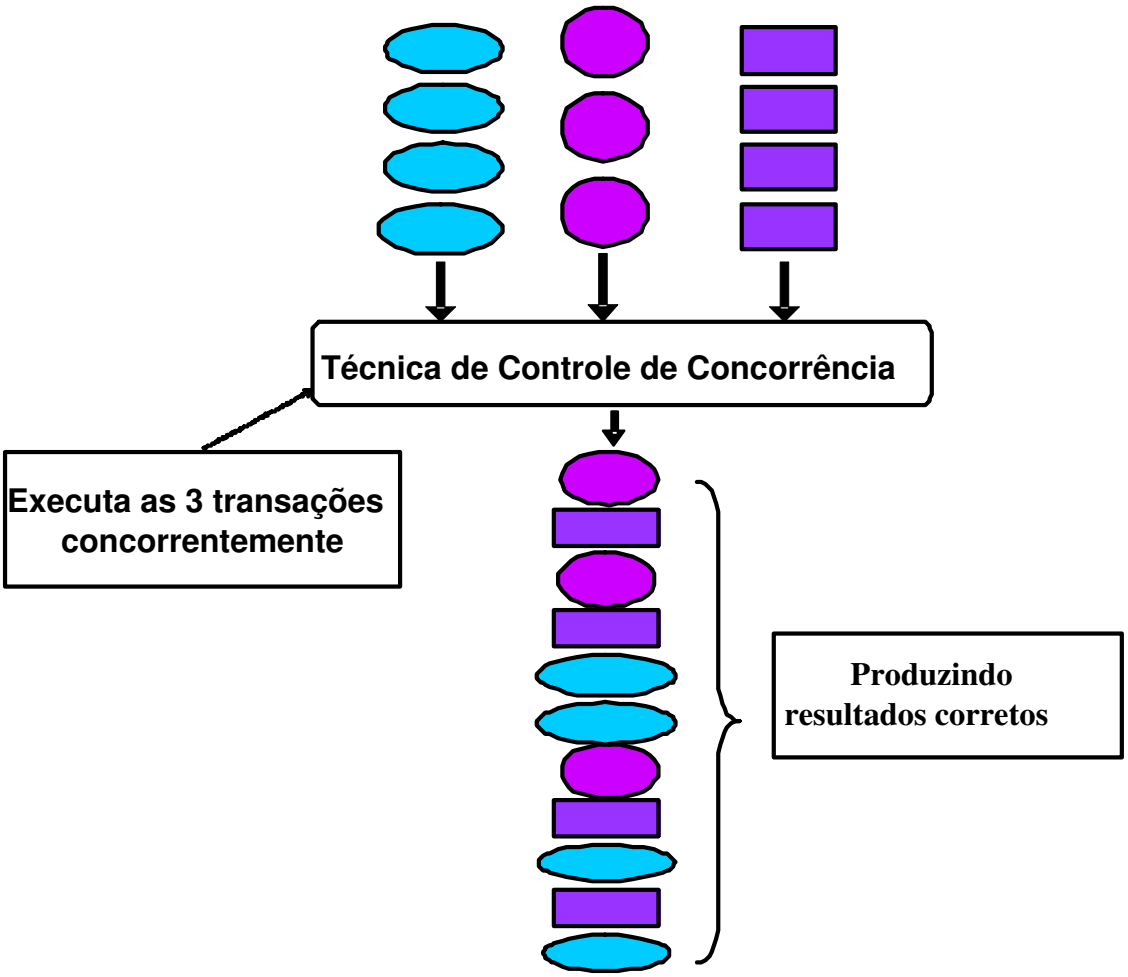

Figura 3.1: Execução Concorrente das Transações

A técnica de controle de concorrência utilizado em um sistema depende do ponto de vista do projetista do sistema. Basicamente podem ser de dois tipos: a otimista e a pessimista.

Na técnica de controle de concorrência pessimista os dados são validados logo ao início do procedimento, como ilustrado na Figura 3.2. Isto impõe ao sistema a necessidade de realizar todo o procedimento de validação dos dados antes do início do processo, supondo que alguma coisa errada esteja para acontecer, ou seja, há um pessimismo relacionada ao procedimento. Portanto, a técnica pessimista deve decidir se aceita ou rejeita uma operação tão logo ele receba esta operação.

Considerando-se que conflito pode ser visto como algo esporádico, ou que tenha uma peridiocidade relativamente alta, certamente o desempenho geral do sistema poderá ser significativamente alterada com uma simples alteração na ordem das fases do processo. Na técnica otimista, como está apresentado na Figura 3.3, a fase de validação ocorre logo

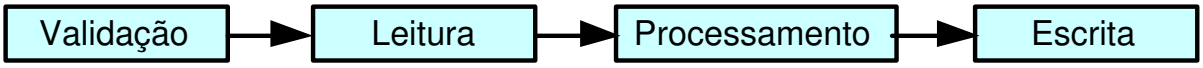

Figura 3.2: Técnica de Controle de Concorrência Pessimista

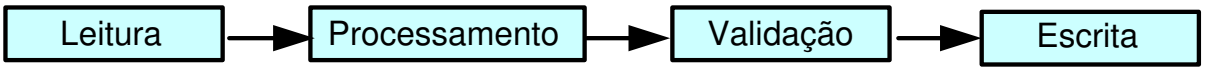

Figura 3.3: Técnica de Controle de Concorrência Otimista

antes ao procedimento de escrita. A fase de validação consiste da conferência de que as alterações em questão mantém a consistência do banco de dados. Desta forma, problemas referentes ao processamento e a leitura abortaram o procedimento antes mesmo de que os dados necessitem de validação.

A maioria dessas técnicas baseia-se no mecanismo de *serialização* como critério de corretude para determinar quais as transações podem executar concorrentemente (SUDARSHAN; KORTH; SILBERSCHATZ, 2001).

A seguir, descreveremos as técnicas utilizadas em um SBGD que são: a técnica de bloqueio de duas fases pessimista e a técnica de controle de concorrência otimista.

3.2.1 Técnica de Bloqueio de Duas Fases Pessimista (2TPL)

A técnica padrão para a solução de problemas de controle de concorrência nos SGBD tradicionais é a 2TPL (HOLANDA; BRAYNER; FIALHO, 2004). Nessa técnica, cada transação emite suas mensagens de solicitação e liberação de bloqueio em duas fases (ELMASRI; NAVATHE, 2002):

- Fase de expansão (ou crescimento): as transações obtém os bloqueios e, nessa fase, não pode existir liberação de bloqueios.
- Fase de encolhimento: as transações liberam os seus bloqueios, no entanto não se pode obter bloqueios nessa fase.

Inicialmente, uma transação está em fase de expansão. A transação adquire os bloqueios necessários. Quando a transação libera um bloqueio, ela entra na fase de encolhimento e não poderá solicitar novos bloqueios.

Essa técnica garante a serialização, porém, não evita impasses [1]. Assim, necessita de uma estratégia para sua detecção, evitando que uma transação fique bloqueada.

[1] *Deadlocks*

3.2.2 Técnica de Controle de Concorrência Otimista (CCO)

Na técnica apresentada anteriormente, a verificação é realizada, antes das operações no BD serem executadas, ou seja, é feita uma verificação de bloqueio no item de dado. Já na CCO, nenhuma verificação é feita enquanto a transação está sendo executada. Existe três fases diferentes que podem ser classificadas nessa técnica (SUDARSHAN; KORTH; SILBERSCHATZ, 2001):

- Fase de leitura: nessa fase se inicia, a execução da transação Ti. Os valores de itens de dados são lidos e armazenados em variáveis locais para Ti. Todas as operações de escrita são processadas com variavéis locais temporárias, sem alterar de fato o banco de dados.

- Fase de validação: A transação Ti processa um teste de validação, para determinar se pode copiar no BD as variavéis temporárias, que mantêm os resultados das operações de escrita sem, com isso, causar a violação da serialização.

- Fase de escrita: Se a transação Ti obtém sucesso na fase de validação, então a atualização é aplicada de fato ao BD. Caso contrário é desfeita.

Cada transação precisa passar pela três fases, na ordem apresentada. Entretanto, as três fases de transaçãoes em execução concorrentes podem ser entrelaçadas.

No entanto, as técnicas de controle de concorrência para SGBD tradicioanis não são apropriados para tratar com as transações de um SGBD-TR, uma vez que as mesmas são caracterizadas por apresentarem restrições de tempo-real que devem ser respeitadas para que obtenha o comportamento temporal desejado ou necessário. Entretanto essas técnicas só consideram aspectos lógicos de dados e transações, desconsiderando os aspectos temporais, necessários aos SGBD-TR.

3.3 Técnica de Controle de Concorrência para SGBD-TR

Em SGBD-TR, as técnicas de controle de concorrência não podem ignorar os requisitos de consistência temporal. Algumas pesquisas tem sido desenvolvidas para técnicas de controle de concorrência em SGBD-TR (SON; LEE, 1994; ALDARMI, 1998; CHIU; KAO; LAM, 1998; LINDSTROM, 2003). Essas técnicas são baseados em: técnica de bloqueio de duas fases pessimista e técnica de controle de concorrência otimista, sendo geralmente extensões das técnicas de controle de concorrência tradicionais. Em seguida serão apresentados alguns trabalhos que utilizam as técnicas de controle de concorrência pessimista e otimista para SGBD-TR.

```
Se pr(Tr) > pr(Th) então
 Tr aborta ou reinicia Th;
Senão
    Trespera;
 Fim
```

Figura 3.4: Algoritmo da T2PL-HP

3.4 Técnica de Bloqueio de Duas Fases Pessimista para SGBD-TR

3.4.1 Técnica de Bloqueio de Duas Fases Pessimista - Alta Prioridade (T2PL-HP)

A técnica de bloqueio de duas fases é mais comum em sistemas de banco de dados convencionais. Com T2PL, a execução da transação consiste em duas fases. Na primeira fase, bloqueios são adquiridas mas podem não ser libertadas. Na segunda fase, bloqueios são libertadas mas podem não ser adquiridas novos bloqueios. No caso de uma transação TR que tem maior prioridade, solicitar um bloqueio que está sendo executado por outra transação, TH, então TR espera, conforme o algoritmo na Figura 3.4. Um problema básico é a possibilidade de inversões de prioridade (CHIU; KAO; LAM, 1998). Uma solução para este problema é reiniciar ou aborta a transação de menor prioridade dando o bloqueio para a transação de maior prioridade, onde conflitos são resolvidos elevando as prioridades das transações.

Todavia, essa técnica pode gerar tempos indeterminados de espera pelo bloqueio comprometendo as restrições de tempo impostas às transações.

3.4.2 Técnica de Bloqueio de Duas Fases Pessimista - Promovido à Espera (T2PL-PE)

Essa técnica possui as características da T2PL-HP, a diferença deste é que o mesmo inclui o mecanismo de herança de prioridade para resolução de conflitos (SON; LEE, 1994). A herança de prioridade foi concebida para resolver o problema de inversão de prioridade. Quando a inversão de prioridade acontece, a transação de menor prioridade que possui o bloqueio executa a prioridade mais alta entre as transações que esperam pelo bloqueio até liberar o bloqueio.

Quando uma transação de baixa prioridade, TH, bloqueia a execução de uma transação de alta prioridade, TR, TR, herda a prioridade de TH e executa como alta prioridade até terminar sua execução, conforme o algoritmo a seguir 3.5.

```
Se   pr(Tr) > pr(Th) então
     Tr espera;
     Th herda a prioridade de   Tr;
   Senão
     Tr espera;
   Fim
```

Figura 3.5: Algoritmo da T2PL-PE

```
TH solicita um item de dado que esta bloqueado por TR
        Se pr(TR)>PR(TH)
          então TH espera por TR;
            senão
             if YR>XH
              então  aborta TR;
            senão TH herda a prioridade de TR;
              fim se
            fim se
```

Figura 3.6: Algoritmo da T2PL-HC

Assim, reduz o tempo de bloqueio das transações de alta prioridade, aumentando a prioridade das transações de baixa prioridade que possui o bloqueio, as transações de baixa prioridade executam mais rapidamente e então liberam o bloqueio mais cedo (LINDSTROM, 2002). Porém, a desvantagem desse técnica é que os tempos de duração de bloqueios das transações de alta prioridade são incertos.

3.4.3 Técnica de Bloqueio de Duas Fases Pessimista - Herança Condicional (T2PL-HC)

A técnica de herança condicional é outra técnica que resolve o problema de inversão de prioridade, integra a política de herança condicional com bloqueios resultando no T2PL-HC. O objetivo desta técnica é minimizar a duração de bloqueios das transações de baixa prioridade e prevenir a formação de impasses.

Se a transação de baixa prioridade estiver próxima de terminar sua execução, herda a prioridade da transação de alta prioridade, assim evitando aborto com desperdício de recursos, se não, a transação de baixa da prioridade é abortada. Desse modo, evita o longo tempo de bloqueio para a transação de alta prioridade, como mostra o algoritmo apresentado 3.6.

3.5 Técnica de Controle de Concorrência Otimista para SGBD-TR

3.5.1 Técnica de Controle de Concorrência Otimista - Transmissão Comprometida (CCO-TC)

Para SGBD-TR, uma extensão da técnica de controle de concorrência tradicional é necessário (LINDSTROM, 2003). Quando uma transação compromete, ela notifica outras transações que estão executando que conflitam com ela, e estas transações conflitantes são reiniciadas imediatamente. Não é necessário verificar conflitos com transações já comprometidas, porque se a transação que esta sendo validada estivesse em conflito com qualquer transação comprometida, então esta teria sido reiniciado antes de chegar a fase de validação.

Isto significa que uma transação quando alcança sua fase de validação é garantida ser comprometida. Essa técnica detecta conflitos mais cedo que a técnica de controle de concorrência tradicional, resultando em reinicio mais cedo, aumentando a possibilidade de encontrar o prazo final das transações.

No entanto, essa técnica apresenta algumas desvantagens. A CCO-TC, não trata com transações longas, como em CCO tradicional, isto porque transações longas são mais prováveis de serem abortadas se existir conflitos. Segundo, como a informação de prioridade não é usada na resolução de conflito, uma transação de prioridade baixa confirmada pode reiniciar uma transação de alta prioridade, quando esta estiver muito perto de sua fase de validação, o que causará perda do prazo final e a transação de alta prioridade é reiniciada.

3.5.2 Técnica de Controle de Concorrência Otimista - Baseado em Sacrifício (CCO-BS)

Modifica a técnica OCC-TC incorporando um mecanismo de sacrifício de prioridade. Quando uma transação alcança sua fase de validação, verifica se existe conflitos com as outras transações que estão executando concorrentemente. Se existir conflitos e pelos menos umas das transações conflitantes tem prioridade alta, então a transação validada é reiniciada, ou seja, sacrifica a transação validada a favor da transação de alta prioridade (ALDARMI, 1998).

Embora esse técnica tenha preferências por transações com alta prioridade, porém apresenta dois problemas (SON; LEE, 1994): (1) se uma transação de alta prioridade causa reinício em uma transação de baixa prioridade, mas falha ao encontrar seu prazo final,

então o reinício foi desnecessário, ou seja, uma transação é sacrificada em favor da outra e esta depois é descartada. Assim diminui o desempenho do sistema; (2) se modificações de prioridade são permitidas, pode haver o problema de reinicio mútuo entre um par de transações, por exemplo, transações serem abortadas. Esses dois problemas apresentados são análogos ao da técnica 2PL-HP.

3.5.3 Técnica de Controle de Concorrência Otimista - Baseado em Espera (COO-BE)

Essa técnica modifica o OCC-TC adicionando um mecanismo de espera de prioridade. Quando uma transação alcança sua fase de validação se sua prioridade não é a mais alta entre as transações conflitantes, então a mesma espera as transações conflitantes de alta prioridade completar sua execução. Essa técnica dar para as transações de alta prioridade a oportunidade para terminar seu prazo final primeiro. Enquanto a transação esta esperando, é possível que a mesma seja reiniciada devido a confirmação de uma transação prioridade alta conflitante.

3.6 Comparativo entre Técnica Pessimista x Técnica Otimista em SGBD-TR

Em SGBD tradicional, a política de resolução de conflito baseado em bloqueio conserva o recurso, enquanto que, na técnica otimista a sua política de resolução de conflito baseado em reinício tender a desperdiçar recursos. Em SGBD-TR, essas técnicas tendem a comportar-se diferentemente. Isto é, a T2PL-HP perde algumas das vantagens de bloqueio da T2PL básico devido a reinício e aborto de transações de baixa prioridade envolvidas em um conflito.

De outra forma, o OCC-TC tende a detectar conflitos mais cedo do que as técnicas de controle de concorrência tradicional, tendo como resultado reinícios mais cedo; assim, menos recursos desperdiçado. Em geral, técnicas baseadas em bloqueios tende a reduzir o grau de paralelismo resultando em escalonamentos seriais; enquanto isso, as técnicas otimistas tentam aumentar o paralelismo ao máximo. O atraso na detecção de conflitos em técnica otimista é realmente vantajosa. Em T2PL-HP, uma transação poderia ser reiniciada ou espera outra transação que será abortada depois. Tal reinicio e espera são inúteis e causa diminuição do desempenho do sistema. Porém, OCC-TC, apenas transações validadas podem causar reinicios de outras transações.

No entanto, OCC apresenta com principal problema em seu desempenho a sobrecarga de reinicio pesada, desperdiçando uma grande quantidade de recursos (CHIU; KAO; LAM,

1998). O OCC-TC reinicia todas as transações conflitantes ativas. O OCC-BS reinicia transações validadas se pelo menos uma transação conflitante tem alta prioridade. O OCC-BE reinicia todas as transações conflitantes ativas. Dessa forma, as técnicas OCC-TC, OCC-BS, OCC-BE apresentam como desvantagem os reinícios desnecessários das transações.

Em suma, as técnicas de controle de concorrência baseadas em bloqueios podem gerar tempos indeterminados de espera por bloqueios. Dessa forma, a utilização desse mecanismo pode ser vantajosa por manter a consistência lógica do banco de dados sem impasses, em contrapartida pode comprometer as restrições de tempo impostas às transações. Por outro lado, na técnica de controle de concorrência otimista, a resolução de conflito é atrasada até a transação está próxima de comprometer, não existe a indeterminação do tempo de espera, porém a quantidade de transações que podem ser reiniciadas é grande, o que pode ser fatal em sistemas com restrições temporais.

Em SGBD-TR, os aspectos temporais dos dados e as restrições de tempo das transações também devem ser considerados. Portanto, um técnica de controle de concorrência para esses sistemas deve também manter a *consistência temporal dos dados e transações* (DIPIPPO, 1995).

Em muitos casos garantir a consistência lógica e temporal simultaneamente para dados e transações pode ser impossível, uma vez que tais exigências são conflitantes em determinadas situações. Então, uma técnica de controle de concorrência deve permitir inconsistência limitada durante a resolução de conflitos, além de permitir a negociação entre a consistência lógica e a consistência temporal dos dados e transações.

Em (DIPIPPO, 1995) foi definido um técnica baseado em informação semântica, denominado *técnica de controle de concorrência semântico* que permite a negociação entre a consistência lógica e temporal dos dados e transações, e limita a imprecisão resultante através de um mecanismo denominado *função de compatibilidade* (FC) (DIPIPPO, 1995). Essa técnica sera apresentada com mais detalhes no próximo Capítulo 4.

Capítulo 4

Técnica de Controle de Concorrência Semântico

Nesse Capítulo é apresentada uma Técnica de Controle de Concorrência Semântico para SGBD-TR. Inicialmente, na Seção 4.1 é apresentado a motivação de utilizar essa Técnica. Em seguida 4.2, é apresentada a Técnica de Controle de Concorrência Semântico para SGBD-TR, sendo o foco do nosso trabalho. Nas Seções 4.3, 4.4 é apresentado os mecanismos em que a Técnica de Controle de Concorrência é baseada que é Serialização *Epsilon* e Função de Compatibilidade, respectivamente. Na Seção 4.5 é apresentado o algoritmo da Técnica de Controle de Concorrência, e finalmente na Seção 4.5.3 a redefinição do algoritmo.

4.1 Introdução

Como já mencionaddo anteriormente, as técnicas de controle de concorrência tradicionais, que utilizam a serialização como critério de corretude, são ineficientes para controlar a concorrência para SGBD-TR. Por esse motivo, diversos pesquisadors propuseram variadas técnicas para solucionar o problema do controle de concorrência em SGBD-TR. Contudo, como mostrado no Capítulo 3, tais propostas apresentam várias desvantagens.

As transações em tempo-real não precisam ser serializáveis, especialmente transações de atualização que gravam informação sobre o mundo real no banco de dados. No entanto, a consequência de relaxar serialização é que uma imprecisão pode ser acumulada no banco de dados e na visão do banco de dados pelas transações. Como por exemplo, uma transação de atualização interrompe uma transação de leitura de modo a manter a consistência temporal dos dados. Nesse caso, a transação de leitura pode ter uma visão imprecisa do dado porque ela pode ler um valor escrito por uma transação de atualização não comprometida. O valor de um item de dado resultante de um escalonamento de tran-

sações é impreciso se ele é diferente do valor correspondente resultante de cada possível escalonamento serializável das mesmas transações.

Neste Capítulo, foi descrito e discutido a respeito de uma técnica de controle de concorrência semântico que utiliza conhecimento especefíco sobre uma aplicação de forma a aumentar a concorrência entre as transações em sistemas de banco de dados.

4.2 Técnica de Controle de Concorrência Semântico

Em (DIPIPPO, 1995) é apresentado uma técnica de controle de concorrência para SGBD-TR baseado em informação semântica. Esta técnica é orientada a objetos e suporta a consistência lógica e temporal dos dados e transações, além de permitir a negociação entre elas quando necessário e controla a imprecisão resultante quando tal negociação é realizada. Essa técnica suporta a consistência temporal das transações por permitir a especificação de escalonamentos mais flexíveis que aqueles permitidos por *serialização* e suas variações (RAMAMRITHAM; PU, 1995).

Para determinar quais transações podem ser executadas concorrentemente esta técnica usa conhecimento semântico a respeito da aplicação (DIPIPPO; WOLFE, 1997). Portanto, visando definir as compatibilidades entre a execução concorrente das transações, as propriedades ACID das transações e a *serialização* precisam ser relaxadas. A conseqüência desse relaxamento, é a geração de uma imprecisão que pode ser acumulada tanto no banco de dados quanto na visão do banco de dados pelas transações. Neste caso, uma transação de leitura pode ter uma visão imprecisa do dado, porque ela pode ler um valor escrito por uma transação de atualização que não foi comprometida.

Nessa técnica o controle de concorrência é distribuído entre os vários objetos da aplicação. Para isso, defini-se FC para cada objeto que controla o acesso concorrente dos métodos ao BD.

4.3 Serialização *Epsilon (Epsilon Serializability - ES)*

Em sistemas em tempo-real, muitas vezes resultados imprecisos podem ser aceitáveis para permitir a satisfação das restrições temporais das transações. Isso impõe às técnicas de controle de concorrência, para tais sistemas, a necessidade de controlar e limitar a imprecisão resultante. Muitas das técnicas para controle de concorrência são baseadas no critério de corretude (*Serialização Epsilon* (RAMAMRITHAM; PU, 1995)).

ES (RAMAMRITHAM; PU, 1995) é um critério de corretude formal que especifica que um escalonamento para transações é correto se o resultado do escalonamento (ambos, valores dos dados e valores dos argumentos de retorno de transações) está dentro de limi-

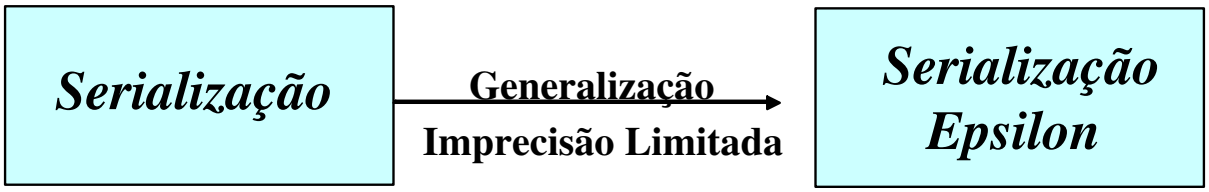

Figura 4.1: Generalização da Serialização Epsilon

tes especificados, ou seja, ele generaliza serialização por permitir imprecisão limitada no processamento de transações, conforme a Figura 4.1. Para acumular e limitar imprecisão, esse critério assume usar apenas dados mensuráveis.

Uma transação especifica a quantidade de imprecisão que ela pode importar e exportar em um item de dado. A quantidade máxima de imprecisão que uma transação t pode importar para um item x é definida como $lim\_imp_{t,x}$, e a quantidade máxima de imprecisão que uma transação t pode exportar para um item x é definida como $lim\_exp_{t,x}$. Para cada item de dados x no banco de dados, lim_max_x é definido como o limite máximo de imprecisão que pode ser escrito em x.

A quantidade de imprecisão importada e exportada por cada transação, assim como as imprecisos escritas nos ítens de dados, devem ser acumuladas durante a execução da transação. A quantidade de imprecisão importada pela transação t para um item de dados x é definida por $imprec\_imp_{t,x}$, e a quantidade de imprecisão exportada pela transação t para um item de dados x é definida por $imprec\_exp_{t,x}$. A quantidade de imprecisão escrita em um item de dados x é definida como $quant\_imp_x$.

As propriedades de segurança definidas para as transações e os dados especificam os limites de imprecisão para transações e dados. Dado um item de dados x, a seguinte propriedade define um dado *seguro*:

$$x : quant\_imp_x \leq lim\_max_x$$

Essa propriedade indica que a imprecisão no item de dados x é aceitável, se ela não for maior que o limite especificado.

Dada uma transação t e um item de dados x, as seguintes propriedades definem uma transação *segura*:

$$(t,x)_{leitura} : imprec\_imp_{t,x} \leq lim\_imp_{t,x}$$

$$(t,x)_{escrita} : imprec\_exp_{t,x} \leq lim\_exp_{t,x}$$

Essa propriedade indica que a imprecisão nos ítens de dados lidos ou escritos por uma transação é aceitável se eles não forem maiores que os limites especificados para eles. Nesse

contexto, dizemos que *ES* está garantida, se e somente se, todos os dados e transações são seguros.

4.4 Função de Compatibilidade (FC)

Uma (FC) é definida pelo projetista do sistema, considerando informações a respeito da aplicação de forma a expressar a compatibilidade entre a execução concorrente dos métodos de um objeto. Desta forma, ela é definida para todo par ordenado de métodos de um objeto e é avaliada em tempo de execução. No entanto, ela só deve ser avaliada quando os métodos estão acessando o mesmo atributo concorrentemente. Em adição, a (FC) acumula a imprecisão introduzida no BD, e controla a imprecisão máxima suportada pelo mesmo. A FC tem a forma:

$$FC(m_{ati}, m_{inv}) = Expressão\ Booleana \Rightarrow IA$$

onde m_{ati} representa o método que está sendo executado e m_{inv} representa o método que foi invocado. A expressão booleana pode conter predicados envolvendo valores dos argumentos dos métodos, do banco de dados e do sistema em geral. onde IA representa a imprecisão acumulada.

4.4.1 Imprecisão Acumulada

Existem três origens potenciais de imprecisão que uma FC pode expressar para as invocações dos métodos m_{ati} e m_{inv} (DIPIPPO, 1995).

1. Imprecisão no valor do argumento de retorno de m_{ati}, quando m_{ati} lê atributos escritos por m_{inv}.
2. Imprecisão no valor do argumento de retorno de m_{inv}, quando m_{inv} lê atributos escritos por m_{ati}.
3. Imprecisão nos atributos que estão no conjunto de atualizações concorrentes de m_{ati} e m_{inv}, isto é, m_{ati} e m_{inv} escrevem nos mesmos atributos.

Quando uma FC é avaliada, se os parâmetros temporais forem avaliados em falso, a transação deve ser abortada, ou esperar em uma fila, caso tenha tempo suficiente para tentar executar novamente. Caso contrário, os parâmetros lógicos serão então avaliados. Quando os parâmetros lógicos são avaliados em verdadeiro o BD poderá ser acessado e a imprecisão gerada (caso exista) será computada. Caso contrário o BD não poderá ser acessado e a transação deverá ser abortada.

Exemplos de Funções de Compatibilidade

Nesta Seção são apresentadas alguns exemplos de FC que expressam compatibilidade condicional para a execução concorrente de métodos.

Exemplo 1

A *FC* 4.1 expressa uma negociação de consistência lógica por consistência temporal quando o método *LerDado* (Ler um dado qualquer) está sendo executado e o método *AtuDado* (Atualizar dado) é invocado. Sob serialização, esses métodos não poderiam ser executados concorrentemente, pois a visão do atributo *dado* pelo método *LerDado* seria prejudicada. No entanto, se a validade temporal do atributo *dado* for violada, é importante permitir a execução do método *AtuDado* para restaurar sua consistência temporal. Porém, os dois métodos só podem ser executados concorrentemente se o valor do atributo *dado* escrito por *AtuDado* ($A.val$) é próximo do valor corrente do atributo *dado* ($dado.val$). Essa determinação é baseada no valor corrente do atributo *dado* ($dado.val$), no limite máximo de imprecisão permitido para o argumento de retorno L de *LerDado* (lim_imp_L), na quantidade de imprecisão que *AtuDado* irá escrever em *dado* através de A ($A.quant\_imp$) e na quantidade de imprecisão existente no argumento de retorno ($L.quant\_imp$). A imprecisão acumulada, resultante da execução concorrente dos dois métodos, também é mostrada. Nesse caso, o argumento de retorno L do método *LerDado* tem um aumento de imprecisão igual à diferença entre o valor do atributo *dado* antes da atualização ($dado.val$) e o novo valor do atributo após a atualização ($A.val$), mais a quantidade de imprecisão que é escrita no atributo *dado* por *AtuDado*, ($A.quant\_imp$).

$$\begin{aligned} FC(LerDado(L), AtuDado(A)) = {} & (agora - dado.tempo \leq dado.avi) \\ & \wedge (|dado.val - A.val| \leq (lim\_imp_L \\ & - (L.quant\_imp + A.quant\_imp))) \\ & \Rightarrow L.quant\_imp = L.quant\_imp \\ & + A.quant\_imp + |dado.val - A.val| \end{aligned} \tag{4.1}$$

Exemplo 2

Na FC 4.2, o método *AtuDado* atualiza o valor do atributo *dado* com o valor do argumento de entrada A e o método *LerDado* lê o valor do atributo *dado*. Esses métodos só poderão ser executados concorrentemente se a diferença entre o novo valor do dado ($A.val$) e o valor original ($dado.val$) estiver dentro do limite de imprecisão aceito pelo argumento de retorno L do método *LerDado* (lim_imp_L). Essa determinação é baseada no valor atualizado por *AtuDado* ($A.val$), no limite de imprecisão permitido para o argumento de

retorno de *LerDado* (lim_imp_L), e na quantidade de imprecisão acumulada no argumento de retorno de *LerDado* ($L.quant\_imp$). A imprecisão acumulada no argumento de retorno L do método *LerDado* tem um aumento igual à diferença entre o valor original ($dado.val$) e o novo valor ($A.val$), caso os métodos sejam executados concorrentemente.

$$\begin{aligned} FC(AtuDado(A), LerDado(L)) = |dado.val - A.val| \leq lim\_imp_L - L.quant\_imp) \\ \Rightarrow L.quant\_imp = L.quant\_imp + |dado.val - A.val| \end{aligned} \tag{4.2}$$

Exemplo 3

A função de compatibilidade 4.3 mostra como um atributo pode se tornar impreciso. Duas invocações de um mesmo método *AtuDado* podem ocorrer concorrentemente se um sensor atualiza o atributo *dado* e um operador humano também solicita a execução de uma operação de escrita nesse mesmo atributo. A função de compatibilidade indica que duas invocações do método *AtuDado* podem ser executadas concorrentemente desde que as diferenças entre os valores escritos, quando executando as duas invocações, não excedam a quantidade de imprecisão permitida para *dado*. A determinação de tal imprecisão é baseada nos valores escritos pelas duas invocações do método ($A_1.val$ e $A_2.val$), na imprecisão existente em *dado* ($dado.quant\_imp$) e no limite de imprecisão permitido para *dado* (lim_max_{dado}). A imprecisão acumulada no atributo *dado* tem um acréscimo igual a diferença entre os valores escritos pelas duas invocações, se os métodos forem executados concorrentemente.

$$\begin{aligned} FC(AtuDado_1(A_1), AtuDado_2(A_2)) = (|A_1.val - A_2.val| \\ \leqslant (lim\_max_{dado} - dado.quant\_imp)) \\ \Rightarrow dado.quant\_imp = dado.quant\_imp \\ + |A_1.val - A_2.val| \end{aligned} \tag{4.3}$$

Para ilustrar o comportamento de uma FC, considere a Figura 4.2 com as seguintes transações:

1. Atualização 1, é uma transação de atualização periódica com prazo firme;
2. Atualização 2, é uma transação de atualização periódica com prazo firme;
3. Transação de leitura 3, é uma transação de usuário periódica com prazo firme;
4. Transação de leitura 4, é uma transação de usuário periódica com prazo firme;

Uma *FC* é avaliada em tempo de execução, considerando inicialmente os parâmetros temporais e em seguida os parâmetros lógicos de cada transação. Considerando o 4.1, quando a Transação de leitura 3 está executando, requisitando uma consulta no atributo temperatura do *Sensor1*, a Transação de Atualização 1 é invocada para atualizar a temperatura adquirida do *Sensor1* no BD. Nestas condições a FC deve ser avaliada. Na FC do 4.2, a Transação de Atualização 2 está atualizando a temperatura do *Sensor2* no BD e a Transação de leitura 4 é invocada para fazer uma consulta da temperatura do *Sensor2*. Portanto, a FC também deve ser avaliada.

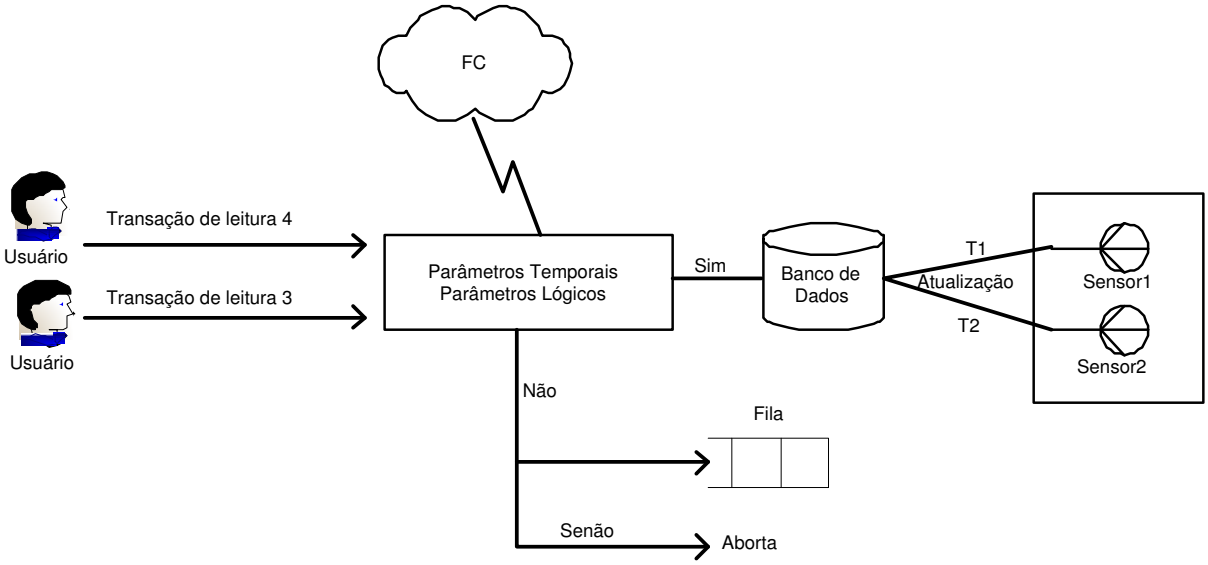

Figura 4.2: Transações possíveis na Rede de Sensores

4.5 Algoritmo da Técnica de Controle de Concorrência Semântico

Nessa seção será apresentado o algoritmo de requisição do bloqueio semântico definido em (DIPIPPO, 1995). Esse algoritmo apresentado em (DIPIPPO, 1995) trata com as ações de *requisição de bloqueio semântico*, *requisição de invocação de método* e *requisição de liberação* de métodos. Em seguida, são apresentados os algoritmos para *requisição de bloqueio semântico* e *requisição de invocação de método* dessa técnica são apresentados nas Seções 4.5.1 e 4.5.2, respectivamente.

4.5.1 Algoritmo do Bloqueio Semântico

Dada uma requisição de bloqueio semântico para m_{inv}, inicialmente a função de compatibilidade $FC(m_{ati}, m_{inv})$ é avaliada para verificar se m_{inv} é compatível com todos os

Algorithm 1 Requisição de Bloqueio Semântico para m_{inv}

1: $Concedido \leftarrow VERDADEIRO$
2: **para todo** $((m_{ati} \in BloqueiosAtivos) \vee ((m_{ati} \in Fila) \wedge (m_{ati}.prio \geqslant m_{inv}.prio)))$ **faça**
3: **se** $FC(m_{ati}, m_{inv})$ **então**
4: Incrementa imprecisão
5: **senão**
6: $Concedido \leftarrow FALSO$
7: **fim se**
8: **fim para**
9: **se** $Concedido$ = FALSE **então**
10: Enfileire(m_{inv}) em $Fila$
11: **senão**
12: Adicione m_{inv} a $BloqueiosAtivos$
13: **fim se**

bloqueios ativos ($BloqueiosAtivos$) e com as requisições de bloqueio que estão na fila de prioridade ($Fila$) com prioridade maior que m_{inv} (Passo 3). A fila de prioridade ($Fila$) é mantida para guardar todas as requisições que não podem ser concedidas imediatamente. Quando a FC é avaliada para *verdadeira*, o mecanismo acumula a imprecisão introduzida com a execução concorrente dos métodos m_{ati} e m_{inv} (Passo 4).

Observe que uma requisição de bloqueio semântico para uma invocação futura de método não contém valores para os argumentos no momento da requisição. Os valores para os argumentos só serão passados no momento da invocação do método. Portanto, quando a função de compatibilidade $FC(m_{ati}, m_{inv})$ é avaliada, se m_{ati} ou m_{inv} são para uma futura invocação de método, então qualquer cláusula da FC que envolve os argumentos do método deve ser avaliada para *falso*.

Quando todos os testes para a função de compatibilidade são avaliados para verdadeiro, m_{inv} é colocado na lista de bloqueios ativos (Passo 12). Quando algum teste falha, m_{inv} é colocado na fila de prioridade ($Fila$) para ser testado novamente quando algum bloqueio for liberado (Passo 10).

4.5.2 Algoritmo da Invocação de Método

Dada uma requisição de invocação de método m_{inv}, o mecanismo inicialmente computa a imprecisão que m_{inv} introduzirá nos atributos que ele escreve (se m_{inv} é um método de escrita) ou em seus argumentos de retorno (se m_{inv} é um método de leitura).

O procedimento de imprecisão inicial (Passo 1) computa a quantidade de imprecisão que m_{inv} escreverá em cada atributo a ($m_{inv}.ImpEscrita(a)$) e que m_{inv} retornará através de cada argumento de retorno r ($m_{inv}.ImpLeitura(r)$). Esses valores são computados usando as quantidades de imprecisão existentes nos atributos ou argumentos de retorno e

calculando como o método atualizará esta imprecisão. Este procedimento pode ser criado pelo projetista do objeto ou por uma ferramenta que examina a estrutura de m_{inv} para determinar como o método irá afetar a imprecisão dos atributos que ele escreve ou retorna.

Algorithm 2 Requisição de Invocação de Método m_{inv}

```
1: ImprecisãoInicial(m_inv)
2: se alguma Precondição falha então
3:    Enfileire m_inv em Fila
4: senão
5:    para todo a ∈ CEA(m_inv) faça
6:       Armazena o valor original de a.quant_imp
7:       a.quant_imp ← m_inv.ImpEscrita(a)
8:    fim para
9:    para todo r ∈ ArgsRetorno(m_inv) faça
10:      Armazena o valor original r.quant_imp
11:      r.quant_imp ← m_inv.ImpLeitura(r)
12:   fim para
13:   se já bloqueado então
14:      Permite a execução de m_inv
15:      Atualiza o Bloqueio Semântico
16:      Verifica Fila
17:   senão
18:      Requisita Bloqueio Semântico
19:      se bloqueio concedido então
20:         Permita a execução de m_inv
21:      senão
22:         para todo a ∈ CEA(m_inv) faça
23:            Recupera o valor original de a.quant_imp
24:         fim para
25:         para todo r ∈ ArgsRetorno(m_inv) faça
26:            Recupera o valor original de r.quant_imp
27:         fim para
28:         para todo argumentos de retorno r de uma invocação de método ativa faça
29:            Recupera o valor original de r.quant_imp
30:         fim para
31:      fim se
32:   fim se
33: fim se
34: Libera Bloqueio
35: Remove m_inv de BloqueiosAtivos
36: Verifica Fila
```

No segundo passo (Passo 2) verificam-se as pré-condições que determinam se a execução de m_{inv} poderia violar a consistência temporal ou limites de imprecisão. As pré-condições são as seguintes:

$$m_{inv}.temporal \Rightarrow \forall a \in CLA(m_{inv}), Agora - a.tempo - Exec(m_{inv}) \leq a.avi \qquad \text{(a)}$$

$$\forall a \in CEA(m_{inv})(m_{inv}.ImpEscrita(a)) \leq lim\_max_a \quad \text{(b)}$$

$$\forall r \in ArgsRetorno(m_{inv})(m_{inv}.ImpLeitura(r)) \leq lim\_imp_r) \quad \text{(c)}$$

A pré-condição (a) assegura que se uma transação requer dados temporalmente válidos, então o método invocado só será executado se os dados que ele lê não perdem sua validade temporal durante a execução do método. A pré-condição (b) assegura que um método de escrita só pode executar se ele não introduz imprecisão além da permitida nos atributos que ele escreve. A pré-condição (c) assegura que um método de leitura só executa se ele não introduz imprecisão além da permitida em seus argumentos de retorno.

Se alguma pré-condição falha, m_{inv} é colocado na fila de prioridade ($Fila$) para ser testado novamente quando um bloqueio for liberado (Passo 3). Se todas as condições forem satisfeitas e m_{inv} é um método de escrita, a quantidade de imprecisão é atualizada para cada atributo a que m_{inv} escreve com o valor $m_{inv}.ImpEscrita(a)$. Se m_{inv} é um método de leitura, a quantidade de imprecisão é atualizada para cada argumento de retorno r de m_{inv} com o valor $m_{inv}.ImpLeitura(r)$ (Passos de 5 a 12). Observe que os valores originais de imprecisão dos atributos e argumentos de retorno envolvidos são salvos, de forma a permitir que eles possam ser recuperados caso o bloqueio não seja concedido.

Observe também que as pré-condições podem bloquear uma transação se os dados que ela acessa são muito imprecisos para seus requisitos. Então deve existir alguma forma de se recuperar a precisão do dado de forma que a transação não fique bloqueada definitivamente. Uma forma é criar transações, que podem ser de um sensor ou de um usuário, para escreverem dados precisos nos dados. Assim as transações bloqueadas pela imprecisão dos dados podem ser executadas.

No passo seguinte (Passo 13), verifica-se se m_{inv} já foi ou não bloqueado. Se não, o bloqueio semântico é requisitado. Caso o bloqueio seja concedido, a execução do método é permitida (Passo 20). Caso contrário, os valores originais de imprecisão que foram trocados são recuperados (Passo 22). Se o bloqueio para m_{inv} foi concedido antes de sua invocação (bloqueio para invocação futura), a execução do método é permitida (Passo 14). Em seguida o mecanismo executa um procedimento de atualização de bloqueio semântico (Passo 15). Este procedimento atualiza o bloqueio associado com m_{inv}, com os respectivos argumentos. Esta atualização é muito importante, pois ela pode aumentar a concorrência entre os métodos, uma vez que uma função de compatibilidade $FC(m_{ati}, m_{inv})$ tem mais chances de ser avaliada para verdadeira quando os argumentos dos métodos estão disponíveis. Após a atualização do bloqueio semântico, as requisições de bloqueio esperando na fila de prioridade ($Fila$) são verificadas para determinar a compatibilidade com o método atualizado (Passo 16).

Finalmente, quando um método termina de executar, o bloqueio correspondente é li-

berado (Passo 34). Quando um bloqueio é liberado ele é removido da lista de bloqueios ativos (Passo 35) e a fila de prioridade é verificada (Passo 36). Como o bloqueio recentemente liberado pode estar associado com a invocação de um método que recuperou a consistência lógica ou temporal de um atributo, ou o bloqueio pode ter causado alguma incompatibilidade, algumas das requisições da fila de prioridade podem ter agora o bloqueio concedido. Também, as requisições de invocação de método da fila de prioridade podem passar nas pré-condições se a consistência temporal ou precisão foi restaurada para o dado. A fila é verificada em ordem de prioridade e, caso algumas dessas requisições sejam concedidas, elas são removidas da fila de prioridade e adicionada na lista de bloqueios ativos.

4.5.3 Algoritmo Redefinido

Uma vez apresentado o algoritmo da técnica de controle de concorrência semântico, identificamos que o algoritmo mostrado é bastante genérico e o propósito do nosso algoritmo e para ser utilizado em aplicações específicas. Portanto, o algoritmo da técnica de controle de concorrência semântico foi redefinido, refinado e implementado de acordo com as necessidades de nossas aplicações.

Para cada par de operações, sejam as operações: $(r_{T1(X)}, w_{T2(X)})$, $(w_{T2(X)}, r_{T1(X)})$, $(w_{T1(X)}, w_{T2(X)})$ pertencentes a diferentes transações $T1$ e $T2$, a FC será avalida de acordo com as seguintes situações descritas:

Quando as operações das transações pertencem a transações diferentes, ou seja, são operações conflitantes;

Quando as operações das transações requisitam a mesma tabela;

Quando as operações das transações requisitam a mesma tupla;

Quando as operações das transações acessam o mesmo atributo

Dessa forma, essas descrições devem ser seguidas quando se definindo uma FC, com o objetivo de se determinar as transações concorrentes e ao mesmo tempo controlar a imprecisão gerada pela execução desses transações. Intuitivamente essas descrições permitem compatibilidade entre duas transações, onde um lê e o outro escreve em um mesmo conjunto de atributos, ou ambos escrevem em um mesmo conjunto de atributos, desde que os limites de imprecisão especificados não sejam violados. A seguir será mostrado os três casos quando a FC é avaliada.

Caso 1. A operação $r_{T1(X)}$ pertencente a transação $T1$ está executando e a operação $w_{T2(X)}$ pertencente a transação $T2$ é invocada para executar, então a FC vai ser avaliada

de acordo com as descrições citadas anteriormente, para ilustrar observe o algoritmo a seguir 3.

Algorithm 3 Algoritmo Redefinido 1

1: *FC verifica se as operações são da mesma transação*
2: *Forem da mesma transação as operações são executadas normalmente*
3: *Forem de diferentes transações*
4: *Verifica se as operações acessam a mesma tabela, tupla e atributo*
5: **se** verdadeiro **então**
6: *Execute FC (leitura, escrita)*
7: *Avalia parâmetros temporais e lógicos*
8: *Concedido em verdadeiro*
9: Incrementa Imprecisão
10: *Concedido em falso*
11: Aborta $w_{T2(X)}$
12: **senão**
13: *A FC não é invocada*
14: **fim se**

Primeiramente verifica-se se as operações das transações são da mesma transação, se forem da mesma transação as operações são executadas normalmente. Se forem de transações diferentes, verifica se ambas operações acessam a mesma tabela, tupla e atributo. Para operações de $r_{T1(X)}$,$w_{T2(X)}$. Se for verdadeiro, então a FC (leitura, escrita) avalia os parâmetros temporais e parâmetros lógicos das operações. Nessa etapa verifica-se se os dados utilizados ainda são válidos temporalmente e se a imprecisão está dentro dos limites de imprecisão especificados. Se o dado é válido temporalmente e os daods estão dentro dos limites de imprecisão, ou seja, se for concedido como verdadeiro, então as duas operações serão executadas concorrentemente e a imprecisão é introduzida no BD, senão $w_{T2(X)}$ é abortado. Se as operações acessam tabelas, atributos e tuplas diferentes, então a FC não é invocada.

È importante observar que, na próxima execução concorrente se as transações acessam o mesmo item de dado , qualquer imprecisão gerada pela execução concorrente das transações deve ser calculada antes da FC ser avaliada. Então a imprecisão que pode ser introduzida é a diferença entre os dois valores que está sendo lido e escrito, $|r_{T1(X)} - w_{T2(X)}|$. Para ser *segura*, essa diferença não pode ser maior que o limite máximo de imprecisão permitida especificado. A imprecisão acumulada é também refletida. A imprecisão acumulada no atributo (X) é igual à imprecisão anterior mais a diferença entre $|r_{T1(X)} - w_{T2(X)}|$.

Caso 2. A operação $w_{T2(X)}$ pertencente a transação $T2$ está executando e a operação

$r_{T1(X)}$ pertencente a transação $T1$ é escalonada para executar concorrentemente com a operação de escrita $w_{T2(X)}$, essas duas operações só poderam ser executadas concorrentes se as mesmas forem compatíveis. Assim foi definido o seguinte algoritmo

Algorithm 4 Algoritmo Redefinido 2

FC verifica se as operações são da mesma transação
Forem da mesma transação as operações são executadas normalmente
3: *Forem de diferentes transações*
Verifica se as operações acessam a mesma tabela, tupla e atributo
se verdadeiro **então**
6: *Execute FC (escrita, leitura)*
Avalia parâmetros temporais e lógicos
Concedido em verdadeiro
9: Incrementa Imprecisão
Concedido em falso
Aborta $r_{T1(X)}$
12: **senão**
A FC não é invocada
fim se

Primeiramente verifica-se se as operações das transações são da mesma transação, se forem da mesma transação as operações são executadas normalmente. Se forem de transações diferentes, verifica se ambas operações acessam a mesma tabela, tupla e atributo. Para operações de $w_{T2(X)}$, $r_{T1(X)}$. Se for verdadeiro, então a FC (escrita, leitura) avalia os parâmetros temporais e parâmetros lógicos das operações. Se for concedido como verdadeiro, então as duas operações serão executadas concorrentemente e a imprecisão é acumulada, senão $r_{T1(X)}$ é abortado. Se as operações acessam tabelas, atributos e tuplas diferentes, então a FC não é invocada.

Caso 3. A operação $w_{T2(X)}$ pertencente a transação $T2$ está executando e a operação ($w_{T1(X)}$ pertencente a transação $T1$ é escalonada para executar concorrentemente com a operação de escrita $w_{T2(X)}$, essas duas operações só poderam ser executadas concorrentes se as mesmas forem compatíveis. Assim foi definido o seguinte algoritmo

Primeiramente verifica-se se as operações das transações são da mesma transação, se forem da mesma transação as operações são executadas normalmente. Se forem de transações diferentes, verifica se ambas operações acessam a mesma tabela, tupla e atributo. Para operações de $w_{T2(X)}$, ($w_{T1(X)}$. Se for verdadeiro, então a FC (escrita, escrita) avalia os parâmetros temporais e parâmetros lógicos das operações, nessa etapa verifica-se se o dado ainda é válido temporalmente. Se for válido temporalmente, ou seja, se for con-

Algorithm 5 Algoritmo Redefinido 3

FC verifica se as operações são da mesma transação
Forem da mesma transação as operações são executadas normalmente
Forem de diferentes transações
Verifica se as operações acessam a mesma tabela, tupla e atributo
se verdadeiro **então**
    Execute FC (escrita, escrita)
    Avalia parâmetros temporais e lógicos
    Concedido em verdadeiro
    Incrementa Imprecisão
    Concedido em falso
    Aborta $w_{T1(X)}$
senão
    A FC não é invocada
fim se

cedido como verdadeiro, então as duas operações serão executadas concorrentemente e a imprecisão é introduzida no BD, senão $w_{T1(X)}$ é abortado.

A imprecisão que pode ser introduzida é a diferença entre os dois valores que está sendo lido e escrito, $|w_{T1(X)} - w_{T2(X)}|$. Para ser *segura*, essa diferença não pode ser maior que o limite máximo de imprecisão permitida especificado. A imprecisão acumulada é também refletida. A imprecisão acumulada no atributo (X) é igual à imprecisão anterior mais a diferença entre $|w_{T1(X)} - w_{T2(X)}|$.

Capítulo 5

Implementação da Técnica de Controle de Concorrência

Nesse Capítulo, é apresentado a implementação da Técnica de Controle de Concorrência enfatizando a Função de Compatibilidade que é base do mesmo. Também é apresentado um estudo de caso focado em Rede de sensores visando validar o método implementado. Inicialmente, na Seção 5.1, é apresentado a descrição da implementação da técnica e, em seguida é apresentado o estudo de caso na Seção 5.2. Finalmente na Seção 5.3 é apresentado a avaliação do desempenho da técnica implementada.

5.1 Descrição da Implementação

As técnicas de controle de concorrência usam escalonadores baseados na *serialização* clássica visando executar as transações concorrentemente enquanto mantém a consistência lógica do banco de dados (RAATIKAINEN; LINDSTROM, 2002). Por outro lado, quando as transações têm restrições temporais, tais como as de um SGBD-TR, os mecanismos precisam utilizar técnicas que garantam tanto a consistência lógica quanto a consistência temporal. Entretanto, existem conflitos entre as restrições lógicas e temporais que precisam ser considerados e gerenciados por tais técnicas.

Neste trabalho foi implementado uma técnica de controle de concorrência baseado em uma extensão da *serialização* clássica denominada (*Serialização Epsilon* (DIPIPPO; WOLFE, 1997). A técnica utiliza uma função de compatibilidade que permite a negociação entre consistência lógica e temporal, além de controlar o grau de imprecisão gerado no banco de dados quando a consistência lógica for sacrificada.

A função de compatibilidade inicialmente considera um par de operações de transações e faz uma análise buscando identificar se existem conflitos entre elas. A análise obedece aos quatro critérios abaixo:

- As operações pertencem à transações distintas,
- Referenciam a mesma tabela,
- Operam na mesma tupla,
- Usam o mesmo atributo

Entretanto, é importante observar que quando o sistema inicia e recebe uma transação com uma única operação, tal análise é desnecessária e a mesma é executada sem restrições.

Se pelo menos um dos requisitos anteriores não for atendido, o método de controle de concorrência executa as operações normalmente.

Se todos os requisitos forem atendidos, a função de compatibilidade faz a análise semântica buscando identificar se as mesmas podem ser executadas concorrentemente ou não. Tal análise é baseada nos tipos das operações (leitura e escrita) e nas informações semânticas contidas nos parâmetros das operações.

Existem quatro tipos de pares de operações possíveis:

1. leitura-leitura
2. leitura-escrita
3. escrita-leitura
4. escrita-escrita

Uma vez determinado o tipo do par, o método apropriado é invocado para executar as operações.

Descrevem-se abaixo as funcionalidades do algoritmo da função de compatibilidade destacando-se os principais métodos. O Apêndice A contém o código desses métodos implementados.

5.1.1 Método (leitura-leitura)

Neste método as operações são executadas na ordem de chegada ao escalonador, ou seja, o banco de dados é acessado pelas duas operações concorrentemente.

5.1.2 Método (leitura-escrita)

Neste método, inicialmente, os parâmetros temporais das operações são verificados visando identificar se os dados são válidos. A verificação é realizada através da seguinte expressão booleana:

((TempoCorrente - TimestampReal) <= AbsoluteValidationInterval).

- Caso a expressão seja avaliada em FALSO, a operação de leitura continua sua execução, no entanto a operação de escrita é devolvida para fila do escalonador para avaliação futura.

- Caso a expressão seja avaliada em VERDADEIRO, calcula-se a imprecisão gerada no sistema: *Imprecisão = ValorOperaçãoEscrita - ValorBancoDeDados*

Observe que a avaliação é feita através da diferença entre o valor que a operação irá escrever e o valor armazenado no banco de dados. Após a imprecisão ser calculada, verifica-se se a mesma é menor ou igual à imprecisão permitida através da expressão: *|Imprecisão| <= ImprecisionPermitida*

Se a expressão acima for avaliada como VERDADEIRO, a imprecisão acumulada é calculada para determinar se a imprecisão permitida não é ultrapassada através da expressão: *(ImprecisãoAcumulada + Imprecisão) <= ImprecisãoPermitida*

- Caso seja avaliada como FALSO, a operação de leitura continua sua execução e a operação de escrita é abortada

- Caso seja avaliada como VERDADEIRO, a execução continua e a Imprecisão é somada a ImprecisãoAcumulada.

Em seguida a imprecisão acumulada é atualizada acrescentando a imprecisão obtida: *ImprecisãoAcumulada = ImprecisãoAcumulada + Imprecisão*

Finalmente, após todas as verificações terem sido avaliadas como verdadeiras, as duas operações podem ser executadas concorrentemente.

5.1.3 Método (escrita-leitura)

Neste método, inicialmente, também os parâmetros temporais das operações são avaliados identificando se os dados são válidos. A verificação é realizada através da seguinte expressão booleana:

((TempoCorrente - TimestampReal) <= AbsoluteValidationInterval))

- Caso a expressão seja avaliada em FALSO, a operação de escrita continua sua execução, no entanto a operação de leitura é devolvida para fila do escalonador para avaliação futura.

- Caso a expressão seja avaliada em VERDADEIRO, calcula-se a imprecisão gerada no sistema: *Imprecisão = ValorOperaçãoEscrita - ValorBancoDeDados*

Observe que a avaliação é feita através da diferença entre o valor que a operação irá escrever e o valor armazenado no banco de dados. Após a imprecisão ser calculada,

verifica-se se a mesma é menor ou igual à imprecisão permitida através da expressão: *|Imprecisão| <= ImprecisionPermitida*

Se a expressão acima for avaliada como VERDADEIRO, a imprecisão acumulada é calculada para determinar se a imprecisão permitida não é ultrapassada através da expressão: *(ImprecisãoAcumulada + Imprecisão) <= ImprecisãoPermitida*

- Caso seja avaliada como FALSO, a operação de escrita continua sua execução e a operação de leitura é abortada.
- Caso seja avaliada como VERDADEIRO, a execução continua e a Imprecisão é somada a ImprecisãoAcumulada.

Em seguida a imprecisão acumulada é atualizada acrescentando a imprecisão obtida: *ImprecisãoAcumulada = ImprecisãoAcumulada + Imprecisão*

Finalmente, após todas as verificações terem sido avaliadas como verdadeiras, as duas operações podem ser executadas concorrentemente.

5.1.4 Método (escrita-escrita)

Neste método, verifica-se, os parâmetros temporais das operações são de forma a identificar se os dados são válidos. A verificação é realizada através da seguinte expressão booleana:

((TempoCorrente - TimestampReal) <= AbsoluteValidationInterval)).

- Caso a expressão seja avaliada em FALSO, a operação de escrita1 continua sua execução, no entanto a operação de escrita2 é devolvida para fila do escalonador para avaliação futura.
- Caso a expressão seja avaliada em VERDADEIRO, calcula-se a imprecisão gerada no sistema: *Imprecisão = ValorOperaçãoEscrita1 - ValorOperaçãoEscrita2*

Observe que a avaliação é feita através da diferença entre o valor da operação de escrita1 e o valor da operação de escrita2. Após a imprecisão ser calculada, verifica-se se a mesma é menor ou igual à imprecisão permitida através da expressão: *|Imprecisão | <= ImprecisionPermitida*

Se a expressão acima for avaliada como VERDADEIRO, a imprecisão acumulada é calculada para determinar se a imprecisão permitida não é ultrapassada através da expressão: *ImprecisãoAcumulada + Imprecisão) <= ImprecisãoPermitida*

- Caso seja avaliada como FALSO, a operação de escrita1 continua sua execução e a operação de escrita2 é abortada.

- Caso seja avaliada como VERDADEIRO, a execução continua e a Imprecisão é somada a ImprecisãoAcumulada.

Em seguida a imprecisão acumulada é atualizada acrescentando a imprecisão obtida: $ImprecisãoAcumulada = ImprecisãoAcumulada + Imprecisão$

Finalmente, após todas as verificações terem sido avaliadas como verdadeiras, as duas operações podem ser executadas concorrentemente.

A técnica implementada foi dividida em três módulos, como ilustrado na Figura 5.1 a seguir:

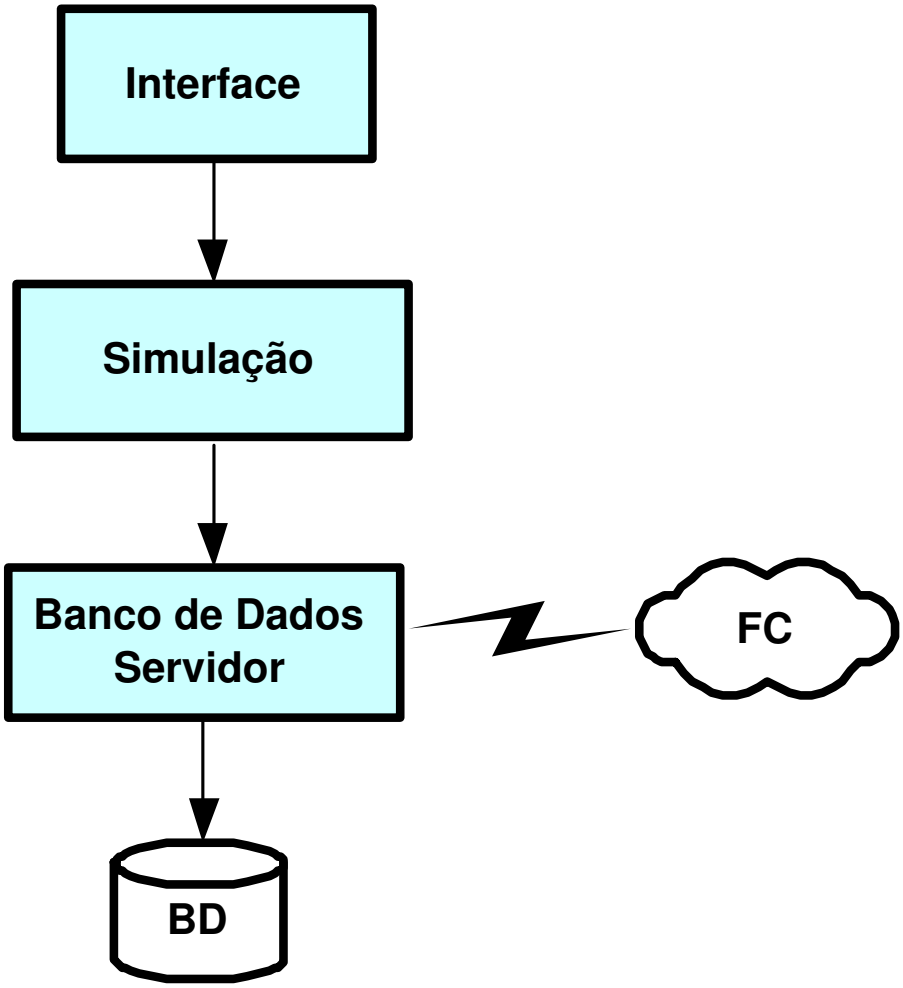

Figura 5.1: Módulos da Implementação

Interface: Inicialmente, foi implementada uma interface, onde o projetista do sistema tem condições de especificar as informações que serão utilizadas na aplicação. Esse módulo é responsável pela interação com o usuário da aplicação através de interface gráfica. Através da interface apresentada na Figura 5.2, temos como parametrizar através de *sensor, timer, min, max, avi, imprec* os dados gerados para ser utilizado na nossa aplicação, onde:

- Sensor: quantidade de sensores usado na simulação;
- Timer: periodicidade do sensor;
- Min: valor inicial da faixa de valores que um dado gerado pelo sensor pode assumir;
- Max: valor final da faixa de valores que um dado gerado pelo sensor pode assumir;
- Avi: prazo de validade absoluta dos dados gerados pelo sensor;
- Imprec: imprecisão permitida limitada

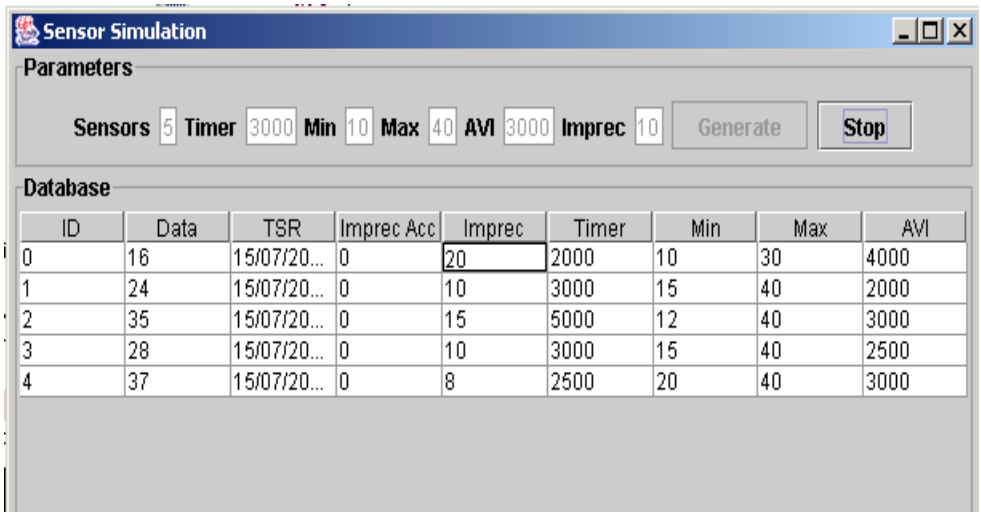

Figura 5.2: Interface Gráfica

Todos os parâmetros utilizados na configuração podem ser definidos de forma global ou individualmente para cada sensor. Além de permitir a configuração da entrada de dados, é possível visualizar as configurações e os valores apresentados por cada um dos sensores. As classes que compõe este pacote são as seguintes:

- *SensorSimulationFrame*: *frame* gráfico que exibe as configurações e os valores apresentados pelos sensores;
- *TableModelSensorSimulation*: coleta e apresenta em uma tabela na interface gráfica os dados gerados em uma simulação.

Simulação: responsável pela simulação do funcionamento dos sensores. Possibilita o disparo e a parada das *threads* que geram valores contidos na faixa determinada pelo usuário através da interface gráfica. As classes que compõe este pacote são as seguintes:

- *Data*: representa os valores apresentados por um sensor durante a execução de uma simulação;
- *ParallellRunnable*: execução simultaneamente duas operações de atualização. Seu método *run* executa o método *start* de ambos os objetos que implementam a interface *Runnable* passados no construtor. Provocando a execução concorrente;
- *Sensor*: simula o funcionamento de um único sensor. Extende a classe *Thread*. Seu método *run* gera valores aleatórios em tempos aleatórios dentro de faixas definidas para os valores e para o tempo;
- *SensorsSimulation*: implementa simulação do funcionamento de sensores. Permite criar simulação com um número determinado de sensores com comportamento configurável;
- *UpdateAction*: *thread* que salva dados gerados por um sensor na simulação.

Banco de Dados Servidor: responsável pelo acesso ao banco de dados e permitindo o acesso de operações conflitantes quanto as mesmas opera sobre um mesmo item de dado no banco de dados. Também inclui código de instrumentação para medição de estatísticas das simulações do funcionamento dos sensores. As classes que compõe este pacote são as seguintes:

- *DatabaseOperation*: classe abstrata que contém os parâmetros de entrada e de saída necessários para a execução de uma operação de leitura ou de atualização de valores de um sensor;
- *ReadOperation*: contém os parâmetros de entrada e de saída necessários para a execução de uma operação de leitura ;
- *UpdateOperation*: contém os parâmetros de entrada e de saída necessários para a execução de uma operação de escrita;
- *DatabaseServer*: responsável pelo acesso aos recursos do banco de dados utilizados pela aplicação. Cria tabelas, insere, remove e consulta dados no banco realizando o mapeamento para objetos da camada de negócios.
- *CompatibilityFunction*: é responsável pela execução concorrente das operações das transações. É nessa classe onde os métodos de (leitura, leitura), (leitura, escrita), (escrita, leitura), (escrita, escrita) são implementados, como apresentado na seção 5.1. Esses métodos só serão invocados quando as operações são de transações diferentes e acessam a mesma tabela, tupla e atributo, como mostra o código 5.3.

```
// Se não são da mesma transação testar se existe colisão...
                                                    else {
                // ...se existe colisão determina que tipo de função de compatibilidade chamar.
                                                    if (op1.collides(op2)) {
                                                // Se as operações são de leitura-leitura...
                                    if ((op1 instanceof ReadOperation) && (op2 instanceof ReadOperation)) {
                                                // ...execute cada uma normalmente na ordem de chegada.
                                        System.out.println("-- Houve colisão mas as operações são de leitura-
                                    leitura.");
                                            CompatibilityFunction.methodReadRead.execute(op1, op2);
                                                    }
                                                // Se as operações são de leitura-escrita...
                                else if ((op1 instanceof ReadOperation) && (op2 instanceof WriteOperation)) {
                                                // Invoca a função de compatibilidade de leitura-escrita.
                                        System.out.println("-- Houve colisão nas operações de leitura-escrita.");
                                            CompatibilityFunction.methodReadWrite.execute(op1, op2);
                                                    }
                                                //Se as operações são de escrita-leitura...
                                else if ((op1 instanceof WriteOperation) && (op2 instanceof ReadOperation)) {
                                                // Invoca a função de compatibilidade para escrita-leitura.
                                        System.out.println("-- Houve colisão nas operações de escrita-leitura.");
                                            CompatibilityFunction.methodWriteRead.execute(op1, op2);
                                                    }
                                                // Se as duas operações são de escrita-escrita...
                                else if ((op1 instanceof WriteOperation) && (op2 instanceof WriteOperation)) {
                                                // Invoca a função de compatibilidade para escrita-escrita.
                                        System.out.println("-- Houve colisão nas operações de escrita-escrita.");
                                            CompatibilityFunction.methodWriteWrite.execute(op1, op2);
                                                    }
                                                    else {
                                                // Algum erro muito sinistro aconteceu...
                                            System.out.println("-- Algum erro muito sinistro aconteceu...");
                                                    }
                                                // Sinaliza que o par de operações foi executado.
                                                    paired = false;
                                                    }
                                            // ...se não existe colisão entre as operações...
                                                    else {
                                                // ...então executa na ordem de chegada.
                                            System.out.println("-- As operações não colidem.");
                                                    op1.execute();
                                                    op2.execute();
                                                    }
                                                }
                                        //Sinaliza que o par de operações foi executado.
                                                    paired = false;
                                                    }
                        // ...se não existe adiciona a operação ao primeiro membro do par.
                                                    else {
                        // Sinaliza que o primeiro membro do par de operações entrou em execução.
                                                    paired = true;
                                            op1 = transaction.nextOperation();
                                                    }
                                                }
```

Figura 5.3: Código da Invocação dos Métodos

5.2 Estudo de Caso

5.2.1 Rede de Sensores

Nos últimos anos, muitas pesquisas têm sido realizadas na área de redes de sensores. Dentre as diversas aplicações para tais redes destacam-se: rastreamento dos produtos estocados em um armazém, monitoramento residencial, detecção de enchente, e monitoramanto de tráfego de veículos (YAO; GEHRKE, 2002, 2003).

As técnicas existentes para o processamento e armazenamento de dados em redes de sensores são insuficientes para as exigências das aplicações citadas acima. Portanto, os SGBD comerciais já estão sendo adaptados para tratar com tais aplicações (TESANOVIC et al., 2002).

Atualmente existem dois tipos de abordagens para o gerenciamento de dados em redes de sensores: A abordagem *warehousing* (BONNET; GEHRKE; SESHADRI, 2001) e a abordagem distribuída (BONNET et al., 1999).

Na abordagem *warehousing*, as redes de sensores são usadas apenas para coletar os dados. O processamento de consultas é executado em dois passos. Primeiro, os dados são extraídos dos sensores e armazenados em um banco de dados servidor. Segundo, o processamento das consultas é realizado diretamente no servidor.

A abordagem distribuída, geralmente usa sensores inteligentes, portanto pode permitir que as consultas sejam processadas tanto no próprio sensor quanto no banco de dados.

A abordagem *warehousing* ainda é bastante adotada devido à capacidade limitada que os sensores convencionais têm de apenas detectar um evento e enviar um sinal, e do alto custo para substituí-los.

Neste trabalho a abordagem *warehousing* foi escolhida para validar e ilustrar as funcionalidades do método de controle de concorrência implementado, considerando que o objetivo da mesma é verificar se os dados estão sendo utilizados enquanto ainda são válidos e se as transações satisfazem seus prazos. Também verifica-se se os limites de imprecisão gerados no banco de dados são respeitados.

5.2.2 Cenário da Aplicação

Para validar o método implementado considera-se uma rede de sensores que é utilizada para monitorar a distância percorrida por um determinado veículo em um sistema de controle de transportes urbanos. Portanto a aplicação precisa ler dos sensores os atributos de tempo e velocidade do veículo para calcular a distância percorrida pelo mesmo. Observe que os atributos de tempo e velocidade devem ser relativamente consistentes de forma que o espaço percorrido seja calculado corretamente. Além disso também é importante considerar uma imprecisão em relação a consistência absoluta entre o valor calculado e armazenado no banco de dados e a nova posição do veículo. No entanto esta imprecisão deve refletir aproximadamente a posição correta do mesmo.

A Figura 5.4, ilustra uma rede de sensores e seus componentes. A mesma é composta por um banco de dados servidor conectado a vários sensores. O banco de dados servidor é atualizado periodicamente através de transações de atualização emitidas pelos sensores e consultados por diversas aplicações.

Os dados em uma rede de sensores possuem um tempo de vida útil limitado. No

caso da aplicação em questão, tal restrição temporal é considerada, tanto no caso da recuperação quanto no processamento dos mesmos.

No entanto é importante ressaltar que algumas aplicações precisam garantir tanto a consistência lógica quanto a consistência temporal do banco de dados. Observe que o método implementado, através da FC permite negociar entre a consistência lógica e temporal a fim de atender ao propósito de diversos tipos de aplicações, além de considerar e controlar a imprecisão gerada no banco.

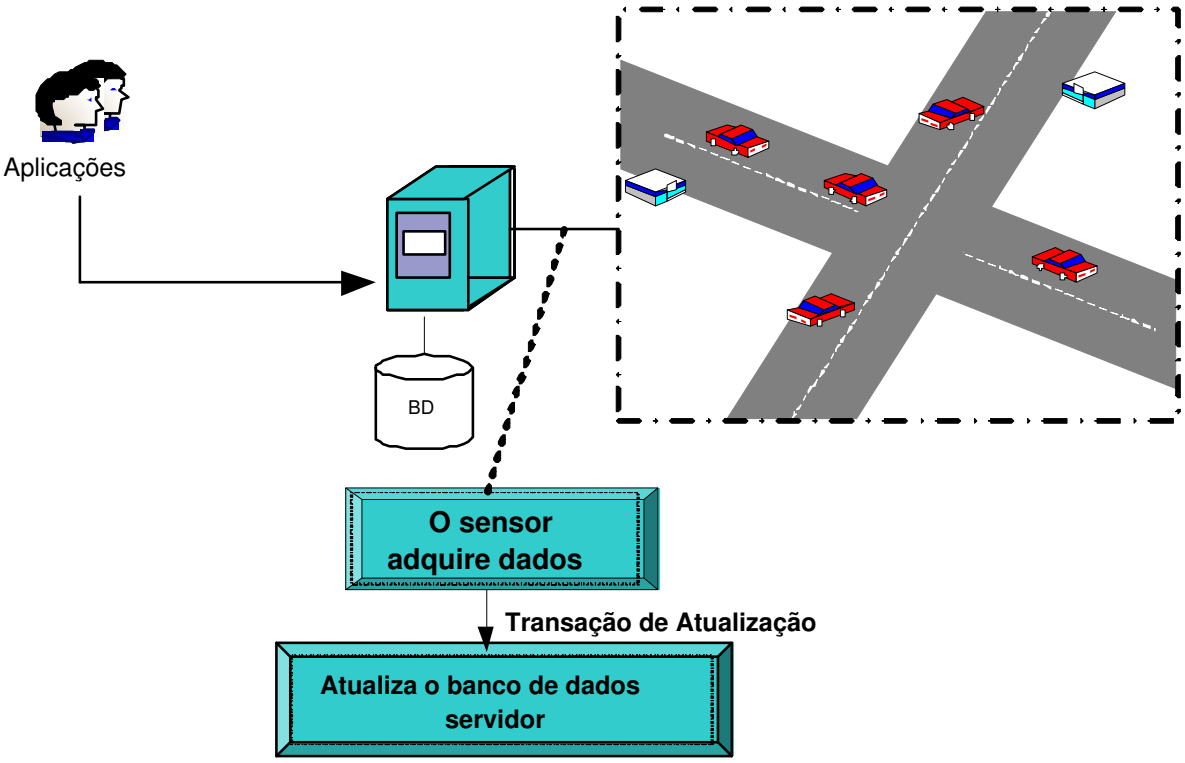

Figura 5.4: Rede de Sensores

Dentre as possíveis transações que devem ser executadas concorrentemente pela aplicação, temos:

1. Transações de Atualização (Sensor)- que adquire dados (tempo e velocidade) dos sensores e escreve no banco de dados;

2. Transações de Consulta (Aplicação)- que lêem dados (tempo e velocidade) do banco de dados.

Considerando uma situação em que o veículo ultrapassou a velocidade permitida (foi definido que a velocidade permitida é de 5Km/h), o sensor é disparado e fotografa a placa do veículo que ultrapassou a velocidade permitida. O banco de dados servidor é atualizado com o dado adquirido do sensor através da transação de atualização. Nesse

instante de tempo a aplicação requisita uma consulta ao banco de dados servidor para verificar as placas dos veículos que foram multados. Esta é uma situação clássica de conflito de transações, onde duas operações, uma de leitura e uma de escrita, pertencentes a transações diferentes tentam acessar o mesmo item de dado, concorrentemente no banco de dados. Essas duas operações só podem executar concorrentemente se a FC for avaliada em VERDADEIRO.

Visando validar a técnica foi criado um banco de dados 5.5 para a aplicação definida, e os métodos implementados foram executados com sucesso. Algumas das situações conflitantes foram consideradas e seus resultados produzidos atenderam os objetivos definidos em 1.3.

ID	v	tsr	avi	ip	ipa	pl
1	30	08:01:30	2000	2	2	Mxz 0906
1	15	08:01:55	3000	13	13	Mmn 7170
2	29	07:59:27	1500	3	3	Hxz 1003
2	24	07:59:25	2000	10	10	Mmx 0014
1	12	08:00:20	2000	2	2	Hxz 5617

Figura 5.5: Tabela Sensor Veículo

5.3 Avaliação do Desempenho da Técnica de Controle de Concorrência

A implementação foi feita em Java(JDK 1.4.1) e o ambiente utilizado foi o Eclipse 3.1. A linguagem de programação Java foi escolhida, uma vez que, disponibiliza bibliotecas e classes necessárias para nossa aplicação e o banco foi armazenado no Sistema de Gerenciamento de Banco de Dados foi o DB2 da IBM.

Foi feita uma simulação onde simulamos através de *threads* transações de leitura fazendo consultas aperiodicamente e transações de atualização (transação de sensor) atualizando o banco de dados periodicamente.

Para analisarmos o desempenho da FC no cenário apresentado em rede de sensores, realizamos simulações entre 50 e 800 operações acessando o BD concorrentemente. Com isso foi possível simular uma situação de concorrência extrema, dificilmente encontrada nos picos de acesso em sistema *Web*, por exemplo, onde os dados coletados por esses sistemas tem uma taxa razoável de acesso, têm picos de acesso que não ultrapassam de 400 usuários por hora(os tempos de execução dos nossos experimentos são de milisegundos).

Foram realizadas várias simulaçãoes onde foi verificado as operações das transações que foram executadas como especificado (lógica e temporalmente), as atualizações e leituras no banco de dados foram executadas com êxito. Na Tabela da Figura 5.6, apresenta as simulações realizadas onde são mostrados os parâmetros das transações e seus respectivos valores. A seguir são apresentadas 8 simulações, visando identificar o desempenho da técnica implementada.

Com a simulação realizada foram obtidos os resultados apresentados nas tabelas qque srão apresentadas a seguir. Na da Figura 5.6, mostra a primeira simulação, onde foram executadas 50 operações normalmente, 6 operações foram executadas com a FC, 4 operações foram abortadas por AVI e 2 operações foram abortadas por IMP.

Primeira Simulação

Parâmetros	Valor	Resultados
Timer	2000 milseg	Operacões executadas : 50 Operações executadas com FC: 6 Operações abortadas por AVI: 4 Operações abortadas por IMP: 2
Min	15	
Max	40	
Avi	500 milseg	
Imp	3	
Impa	3	

Figura 5.6: Parâmetros das Transações da Primeira Simulação

Em uma segunda simulação, como apresentado na tabela da Figura 5.7, com os respectivos parâmetros das transações e os valores, foram executadas 150 operações, onde a FC foi avaliada durante a execução de 26 operações, 14 operações foram abortadas por AVI e 2 operações foram abortadas por IMP.

Segunda Simulação

Parâmetros	Valor	Resultados
Timer	2000 milseg	Operacões executadas :150 Operações executadas com FC:26 Operações abortadas por AVI: 14 Operações abortadas por IMP: 2
Min	15	
Max	40	
Avi	1000 milseg	
Imp	5	
Impa	5	

Figura 5.7: Parâmetros das Transações da Segunda Simulação

Na terceira simulação, como apresentado na tabela da Figura 5.8, com os respectivos parâmetros das transações e os valores, foram executadas 250 operações, onde a FC foi

avaliada durante a execução de 52 operações, onde 25 operações foram abortadas por AVI e 8 operações foram abortadas por IMP.

Terceira Simulação		
Parâmetros	Valor	Resultados
Timer	3000 milseg	Operacões executadas :250 Operações executadas com FC:52 Operações abortadas por AVI: 25 Operações abortadas por IMP: 8
Min	15	
Max	40	
Avi	1500 milseg	
Imp	8	
Impa	8	

Figura 5.8: Parâmetros das Transações da Terceira Simulação

Na quarta simulação, como apresentado na tabela da Figura 5.9, com os respectivos parâmetros das transações e os valores, foram executadas 350 operações, onde a FC foi avaliada durante a execução de 70 operações, onde 29 operações foram abortadas por AVI e 12 operações foram abortadas por IMP.

Quarta Simulação		
Parâmetros	Valor	Resultados
Timer	2000 milseg	Operacões executadas :350 Operações executadas com FC:70 Operações abortadas por AVI: 29 Operações abortadas por IMP: 12
Min	15	
Max	40	
Avi	2000 milseg	
Imp	11	
Impa	11	

Figura 5.9: Parâmetros das Transações da Quarta Simulação

Na quinta simulação, como apresentado na tabela da Figura 5.10, com os respectivos parâmetros das transações e os valores, foram executadas 450 operações, onde a FC foi avaliada durante a execução de 88 operações, onde 17 operações foram abortadas por AVI e 9 operações foram abortadas por IMP. A partir da quinta simulação observou-se que a quantidade de operações que foram abortadas por AVI e IMP diminuiram, com isso o desempenho da técnica implementada tende a aumentar.

Na sexta simulação, como apresentado na tabela da Figura 5.11, com os respectivos parâmetros das transações e os valores, foram executadas 550 operações, onde a FC foi avaliada durante a execução de 151 operações, onde 4 operações foram abortadas por AVI e 8 operações foram abortadas por IMP.

Quinta Simulação		
Parâmetros	Valor	Resultados
Timer	2000 milseg	Operacões executadas : 450 Operações executadas com FC: 88 Operações abortadas por AVI: 17 Operações abortadas por IMP: 9
Min	15	
Max	40	
Avi	2500 milseg	
Imp	15	
Impa	15	

Figura 5.10: Parâmetros das Transações da Quinta Simulação

Sexta Simulação		
Parâmetros	Valor	Resultados
Timer	2000 milseg	Operacões executadas : 550 Operações executadas com FC:151 Operações abortadas por AVI: 4 Operações abortadas por IMP: 8
Min	15	
Max	40	
Avi	3000 milseg	
Imp	18	
Impa	15	

Figura 5.11: Parâmetros das Transações da Sexta Simulação

Na sétima simulação, como apresentado na tabela da Figura 5.12, com os respectivos parâmetros das transações e os valores, foram executadas 650 operações, onde a FC foi avaliada durante a execução de 196 operações, onde 6 operações foram abortadas por AVI e 8 operações foram abortadas por IMP.

Sétima Simulação		
Parâmetros	Valor	Resultados
Timer	2000 milseg	Operacões executadas : 650 Operações executadas com FC:196 Operações abortadas por AVI: 6 Operações abortadas por IMP: 8
Min	15	
Max	40	
Avi	3500 milseg	
Imp	20	
Impa	20	

Figura 5.12: Parâmetros das Transações da Sétima Simulação

Na oitava simulação, como apresentado na tabela da Figura 5.13, com os respectivos parâmetros das transações e os valores, foram executadas 750 operações, onde a FC foi

avaliada durante a execução de 228 operações, onde 1 operações foram abortadas por AVI e 1 operações foram abortadas por IMP.

Oitava Simulação		
Parâmetros	Valor	Resultados
Timer	2000 milseg	Operacões executadas : 750 Operações executadas com FC:228 Operações abortadas por AVI: 1 Operações abortadas por IMP: 1
Min	15	
Max	40	
Avi	4000 milseg	
Imp	23	
Impa	23	

Figura 5.13: Parâmetros das Transações da Oitava Simulação

Considerando os resultados das simulações com as respectivas características das operações das transações, foi necessário verifica através de gráficos o desempenho da técnica de controle de concorrência implementada, a fim de verificarmos as propriedades lógicas e temporais das transações. Na Figura 5.14 ilustra a execução das operações das transações, onde mostra a quantidade de operações que formam executadas com a FC, como também as operações que foram executadas normalmente sem a FC.

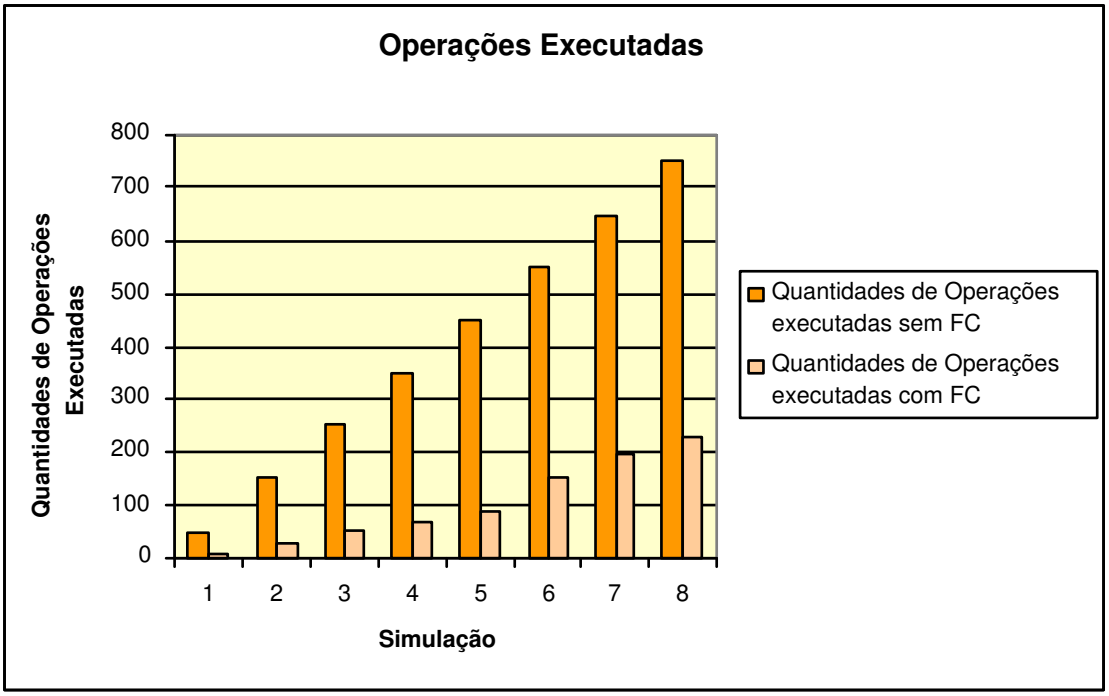

Figura 5.14: Operações Executadas

Em seguida nas Figuras 5.15 e 5.16 são apresentados os gráficos onde mostram as operações que foram abortadas por o dado ter perdido sua validade absoluta (AVI) como

também as operações que foram abortadas porque não estavam dentro dos limites de imprecisão permitida (IMP), é importante observar que os gráficos apresentados também identificam a quantidade de operações que formam executadas concorrentemente com a avaliação da FC.

O intervalo de validade absoluta do dado é relacionado diretamente à consistência temporal dos dados, porque é este intervalo que define a consistência temporal dos dados. Assim, foram feitas simulações com a técnica implementada executando com os valores variando validade absoluta, os testes foram realizados usando os valores entre (500 a 4000 milesegundos). Com relação a imprecisão permitada pelo dado, nas três primeiras simulações foram utilizados valores médios entre 3 a 8, da quarta a oitava simulação os valores foram mais altos entre 10 a 23. Os resultados das simulações indicam que a técnica implementada permite maior concorrência e fornece mais suporte para encontrar as restrições de tempo das transações.

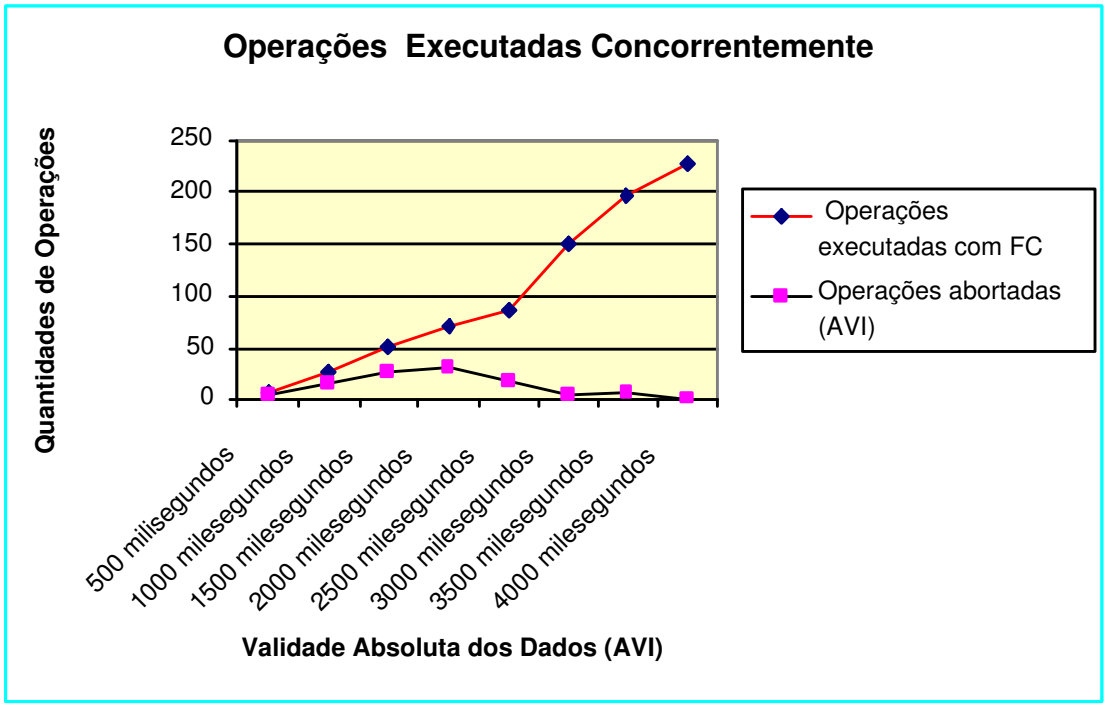

Figura 5.15: Operações Executadas Concorrentemente e Operações Abortadas (AVI)

Com base nos gráficos mostrados, observa-se um aumento considerável de execuções concorrentes, mantendo a consistência do BD. Esse aumento é em decorrência dos limites de imprecisão ser definidos em tempo de execução de acordo com a aplicação.

Foi verificado também um grau de paralelismo entre as transações é maior utilizando o critério de corretude serialização *epsilon*, onde o mesmo tem a finalidade de introduzir uma maior concorrência entre as transações, uma vez que, relaxa a serialização. Dessa forma, diminui o número de transações abortadas e aumenta a eficiência da técnica de

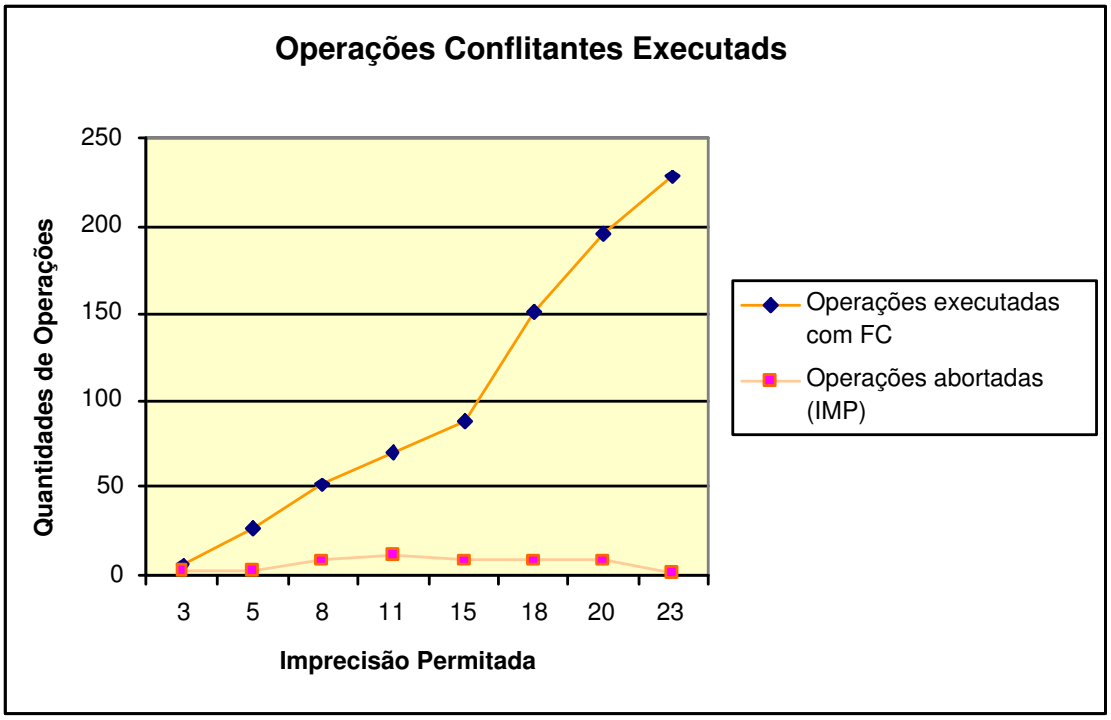

Figura 5.16: Operações Executadas Concorrentemente e Operações Abortadas (IMP)

controle de concorrência semântico. Já que é obtido informação sobre o estado corrente do objeto e parâmetros atuais das operações.

Também foi feito uma análise com relação ao desempenho das técnicas de controle de concorrência existentes para SGBD-TR e observou-se que a técnica implementada executou melhor do que as outras. No capítulo 3 foram apontadas várias técnicas de controle de concorrência e foi indicado porque cada uma das técnicas descritas não se enquadravam com os objetivos do nosso trabalho. Uma vez que, essas técnicas não consideram nenhuma informação a respeito dos dados, de forma a determinar como as transações podem ser executadas concorrentemente. As técnicas baseadas em bloqueios apresentam o problema de inversão de prioridade, já as técnicas de controle de concorrência otimista é baseada em reinícios o que pode ser fatal em sistemas com restrições temporais, podendo comprometer as restrições de tempo impostas às transações.

Portanto, a grande motivação para utilizar a técnica de controle de concorrência baseado em informação semântica é permitir uma imprecisão no valor do item de dado é a de atender as restrições de tempo-real impostas aos dados e transações nos SGBD-TR.

Capítulo 6

Conclusão

O objetivo principal deste trabalho foi implementar uma técnica de controle de concorrência para SGBD-TR, denominada Técnica de Controle de Concorrência Semântico. A técnica é baseada em uma extensão da serialização clássica, denominada (*Serialização Epsilon* (RAMAMRITHAM; PU, 1995), e em uma função de compatibilidade (DIPIPPO, 1995). Ela permite o gerenciamento da execução concorrente de transações com restrições temporais, pois tais transações geralmente manipulam dados com tempo de vida útil limitado, portanto precisam ser utilizados enquanto são válidos. Como não é possível garantir a consistência lógica e temporal simultaneamente no banco de dados, ela permite a negociação entre ambas quando necessário através da introdução de uma imprecisão no banco de dados. No entanto, a imprecisão resultante desta negociação é controlada.

A técnica foi implementada através da linguagem de programação Java e do SGBD DB2 da IBM (CORPORATION, 2004). É importante destacar que apesar do SGBD utilizado ter sido o DB2 da IBM a técnica pode ser utilizada com qualquer SGBD, desde que, a técnica de controle de concorrência do mesmo seja desativada, assim como foi feito neste trabalho.

Visando validar a técnica implementada foi desenvolvida uma aplicação para rede de sensores com seu respectivo banco de dados. Através dela verificou-se que a técnica implementada foi executada com sucesso. Algumas das situações conflitantes foram consideradas e os resultados produzidos atenderam os objetivos definidos em 1.3.

6.1 Contribuições

A técnica de controle de concorrência implementada pode ser utilizada em qualquer SGBD-TR, ou inserida diretamente em qualquer aplicação Java que precise tratar com dados e transações com restrições temporais sem a necessidade da aquisição de um SGBD-TR. Portanto, destacam-se duas contribuições principais deste trabalho:

- A implementação e validação de uma técnica de controle de concorrência semântico para SGBD-TR;
- A disponibilização desta técnica (através de classes Java) para ser utilizada em qualquer aplicação Java que precise tratar com dados e transações com restrições temporais sem a necessidade da aquisição de um SGBD-TR.

6.2 Perspectivas Futuras

A atual implementação da técnica de controle de concorrência semântico contempla as funcionalidades desejadas para aplicações que precisem tratar com restrições temporais *soft* e *firm*, no entanto, não garante restrições temporais do tipo *hard*. Portanto, destacam-se a seguir, alguns tópicos de pesquisa que precisam ser realizados para que o mesmo possa garantir também tais restrições e ser integrado a um SGBD-TR:

1. Implementar esta técnica utilizando um sistema operacional em tempo-real, tais como RT/Linux, RTAI, KNX, entre outros;
2. Implementar esta técnica utilizando a Especificação Java para Tempo-Real (SUN MICROSYSTEMS, 2003), utilizando a máquina virtual do Jamaica (JAMAICAVM, 2005). Esta implementação permitirá a avaliação do desempenho da técnica, comparando-se as duas implementações;
3. Integrar com a interface para uma linguagem de consulta de tempo-real desenvolvida em (LEITE et al., 2005), com a finalidade de verificar a execução concorrente das consultas elaboradas através da mesma;
4. Transformar as consultas elaboradas na linguagem LC-BDTR (LEITE et al., 2005), em operações de leitura e escrita, reconhecidas pela técnica implementada. Tal implementação já está sendo realizada;
5. Validar a técnica do controle de concorrência através de aplicações para rede de sensores inteligentes.

Referências Bibliográficas

AL-OMARI, R.; SOMANI, K. A.; MANIMARAN, G. Efficient overloading techniques for primary-backup scheduling in real-time systems. *Journal of Parallel and Distributed Computing*, Iowa State University, March 2004.

ALDARMI, A. S. Real-time databases systems: Concepts and design. University of York, April 1998.

BESTRAVOS, A.; LIN, K.-J.; S.H., S. *Real-Time Database Systems: Issues and Applications*. Boston: Kluwer Academic Publishers, 1997.

BONNET, P. et al. *Query Processing in a Device Database Systems*. [S.l.], Outubro 1999.

BONNET, P.; GEHRKE, J.; SESHADRI, P. Towards Sensor Database Systems. *2nd International Conference on Mobile Data Management*, p. 3–14, Janeiro 2001. Hong Kong.

CHIU, A.; KAO, B.; LAM, K. An analysis of lock-based and optimistic concurrency control protocols in multiprocessor real-time databases. *Journal of Systems and Software*, University of Hong Kong, 1998.

CORPORATION, I. Db2 universal database. Http://www.ibm.com/db2. November 2004. Disponível em: <http://www.ibm.com/db2>.

DEITEL, H.; DEITEL, P. *Java: Como Programar. 3.edição*. São Paulo: Bookman Companhia Ed., 2000.

DEITEL, H.; DEITEL, P. *Java: Como Programar. 4.edição*. São Paulo: Bookman Companhia Ed., 2002. ISBN 85-363-0123-6.

DIBBLE, C. P. *Real-Time Java Platform Programming. 1. Edition*. [S.l.]: Prentice Hall, 2002. 352 p. ISBN 0130282618.

DIPIPPO, L. *Semantic Real-Time Object-based Concurrency Control*. Tese (Doutorado) — Department of Computer Science and Statistics, University of Rhode Island, 1995.

DIPIPPO, L.; WOLFE, V. *Object-Based Semantic Real-Time Concurrency Control with Bounded Imprecision*. [S.l.], 1997.

DIPIPPO, L.; WOLFE, V. *Performance of Object-Based Semantic Real-Time Concurrency Control*. [S.l.], 1997.

ELMASRI, R.; NAVATHE, S. *Sistemas de banco de dados: fundamentos e aplicações. 3. edição*. Rio de Janeiro: Addison–Wesley Pub. Co., 2002.

FARINES, M. R.; FRAGA, S. F.; OLIVEIRA, S. R. *Sistemas em Tempo-Real*. [S.l.]: 12ª Escola de Computação, 2000. 4-13 p.

FERNANDES, Y. et al. Implementation of transactions scheduling for real-time databases management. *IEEE Systems, Man and Cybernetics Conference*, 2004.

FILHO, M. *Teste do Grafo de Serialização Temporal:Uma Estratégia para o Controle de Concorrência em Ambientes Broaadcast*. Dissertação (Mestrado), Universidade Federal do Ceará, Outubro 2001.

GRAHAM, H. M. How to get serializability for real-time transactions without having to pay for it. *In Proceedings of the XIV IEEE Real-Time Systems Symposium*, p. 56–65, 1993.

HOLANDA, M.; BRAYNER, A.; FIALHO, V. Controle de concorrência em banco de dados para ambientes de computação móvel. *VI Workshop de Comunicação Sem Fio e Computação Móvel*, p. 161–170, 2004.

HUNG, V.; HUONG, V. *Modelling Real-time Database Systems in Duration Calculus*. [S.l.], August 2002.

JAMAICAVM. Available in web, http://www.aicas.com. April 2005.

JENSEN, S. C. *Introduction to Temporal Database Research*. Tese (Tese de Doutorado) — , abr. 2000.

KANG, K.; SON, S. H.; STANKOVIC, J. Managing Deadline Miss Ratio and Sensor Data Freshness in Real-Time Databases. *IEEE Transactions on Knowledge and Data Engineering*, v. 16, n. 07, July 2004.

KANG, K.-D. *qRTDB: QoS-Sensitive Real-Time Database*. Tese (Doutorado) — Department of Computer Science, University of Virginia, December 2001.

LAM, K.-Y. et al. Evaluation of concurrency control strategies for mixed soft real-time database systems. *Inf. Syst.*, Elsevier Science Ltd., v. 27, n. 2, p. 123–149, 2002. ISSN 0306-4379.

LEA, D. *Concurrent Programming in Java: Design Principles and Patterns*. Second. [S.l.]: Addison–Wesley, 1999. ISBN 0-201-31009-0.

LEITE, C. M. et al. Ql-rtdb: Query language for real-time database. *Anais 7th International Conference on Enterprise Information Systems*, p. 04, Maio 2005.

LINDSTROM, J. Integrated and adaptive optimistic concurrency control method for real-time databases. *VIII International Conference on Real-Time Computing Systems and Application*, University of Helsinki Finland, 2002. VIII International Conference on Real-Time Computing Systems and Application.

LINDSTROM, J. *Optimistic Concurrency Control Methods for Real-Time Database*. Tese (Doutorado) — Department of Computer Science, University of Helsinki Finland, January 2003.

LIU, J. W. S. *Real-Time Systems*. 1. ed. 2000.

MACEDO, R. et al. Tratando a previsibilidade em sistemas de tempo-real distribuídos: Especificação, linguagens, middleware e mecanismos básicos. In: _____. Universidade Federal da Bahia: [s.n.], 2004. p. 15–20.

NETO, P. R. et al. Uma aplicação de bancos de dados em tempo-real para redes de sensores. *VI Workshop de Tempo Real(WTR) - SBRC*, p. 45–52, 2004.

NYSTRöM, D. et al. Pessimistic concurrency control and versioning to support database pointers in real-time databases. In: *Proceedings of Euromicro conference on Real-Time Systems (ECRTS)*. [S.l.: s.n.], 2004.

PERKUSICH, M. *Um Método Baseado em Redes de Petri para a Modelagem de Bancos de Dados para Aplicações em Tempo-Real*. Tese (Tese de Doutorado) — Curso de Doutorado em Engenharia Elétrica, CCT/UFPB, abr. 2000.

RAATIKAINEN, K.; LINDSTROM, J. Using real-time serializability and optimistic concurrency control in firm real-time databases. University of Helsinki Finland, 2002.

RAMAMRITHAM, K.; PU, C. A formal characterization of epsilon serializability. *IEEE Transactions on Data and Knowledge Engineering*, v. 6, n. 7, p. 997–1007, 1995.

RAMAMRITHMAN, K. *Real-Time Databases*. 1993.

RAMAMRITHMAN, K.; SON, H. S.; DIPIPPO, L. Real-time databases and data services. In: *Real-Time Systems*. Indian Institute of Technology: Kluwer Academic Publishers, 2004. p. 179–215.

RIBEIRO NETO, P. *Análise de Controle de Concorrência e Escalonamento em Bancos de Dados em Tempo-Real Usando Redes de Petri Coloridas*. Dissertação de Mestrado, Universidade Federal da Paraíba, Campina Grande, PB: [s.n.], Agosto 2001.

RIBEIRO NETO, P. *Uma Especificação Formal de Seriação Epsilon considerando propriedades de QoS*. Universidade Federal de Campina Grande, PB, Maio 2003.

RIBEIRO NETO, P. *Gerenciamento de Qualidade de Serviços para Banco de Dados em Tempo-real*. Tese (Exame de Qualificação) — Curso de Doutorado em Engenharia Elétrica, Universidade Federal da Paraíba, Campina Grande, PB, 2005.

RIBEIRO NETO, P.; PERKUSICH, A.; M.L.B., P. Modelagem e Análise de Qualidade de Serviços para Bancos de Dados em Tempo-Real. *V Workshop de Tempo Real*, p. 63–70, 2003.

RIBEIRO NETO, P.; PERKUSICH, M.; PERKUSICH, A. Formal Specification of the Epsilon Seriazibility Considering Quality of Service. *IEEE Systems, Man and Cybernetics Conference*, 4, p. 4027 – 4032 , 2003.

RIBEIRO NETO, P.; PERKUSICH, M.; PERKUSICH, A. Real-Time Database Modeling Considering Quality of Service. *5th International Conference on Enterprise Information Systems*, 3, p. 403–410, 2003.

RIBEIRO NETO, P. et al. A Model for Real-Time Databases. *Argentine Symposium on Software Engineering*, 2004.

SON, H. S.; LEE, J. Performance of concurrency control algorithms for real-time databases systems. University of Virginia, p. 56–65, April 1994.

SQUADRITO, M. A. *Extending the Priority Ceiling Protocol Using Read/Write Affected Sets*. Dissertação (Mestrado) — Department of Computer Science, University of Rhode Island, 1996.

STANKOVIC, J.; SON, S.; HANSSON, J. Misconceptions about real-time databases. *IEEE Computer*, v. 32, n. 6, p. 29–36, 1999.

SUDARSHAN, S.; KORTH, F. H.; SILBERSCHATZ, A. *Database System Concepts*. 4. ed. 2001. 840 p.

SUN MICROSYSTEMS. *The Real Time Specification for Java*. [S.l.], Acesso em Março 2003. Http://rtsj.dev.java.net.

TESANOVIC, A. et al. *Embedded databases for embedded real-time systems: A component-based approach*. [S.l.], 2002.

YAO, Y.; GEHRKE, J. E. The Cougar Approach to In-Network Query Processing in Sensor Networks. *Sigmod Record*, 31, n. 3, Setembro 2002.

YAO, Y.; GEHRKE, J. E. Query Processing for Sensor Networks. *To appear in the First Biennial Conference on Innovative Data Systems Research (CIDR 2003)*, Janeiro 2003. Asilomar, California.

Apêndice A

Código da Técnica de Controle de Concorrência

A.1 Classe *CompatibilityFunction*

```
package rtdbms.protocol;

import rtdbms.operation.OperationAdapter;

public interface CompatibilityFunctionInterface {

public void executeReadRead(OperationAdapter op1, OperationAdapter
op2);
public void executeReadWrite(OperationAdapter op1, OperationAdapter
op2);
public void executeWriteRead(OperationAdapter op1, OperationAdapter
op2);
public void executeWriteWrite(OperationAdapter op1, OperationAdapter
op2);
}
```

A.2 Classe *MethodReadRead*

```
package rtdbms.protocol;
```

```
import rtdbms.operation.OperationInterface;

public class MethodReadRead {

  public MethodReadRead() {
  super();

}

public void execute(OperationInterface opread1, OperationInterface
opread2) {

System.out.println("CF (R,R) Invocada");
        opread1.execute();
        opread2.execute();
System.out.println("FINAL: " + opread1.toString() + ", " +
opread2.toString());
    }

}
```

A.3 Classe *MethodReadWrite*

```
package rtdbms.protocol;

import java.util.Date;

import rtdbms.operation.OperationAdapter;

public class MethodReadWrite {

public MethodReadWrite() {}

public void execute(OperationAdapter opread, OperationAdapter
opwrite) {
```

```
 long timestampReal = opread.table.atp.timestampReal;
 long absoluteValidationInterval =
 opread.table.atp.absoluteValidationInterval;
 long allowedImprecision =
 opread.table.alp.allowedImprecision;
 long accumulatedAllowedImprecision =
 opwrite.table.alp.accumulatedAllowedImprecision;

 System.out.println("CF (R,W) Invocada");

// VALIDAÇÃO DOS PARÂMETROS TEMPORAIS

// Tempo atual do sistema em milisegungos.
long currentTime = new Date().getTime();

// A diferença de tempo em milisegundos é obtida pela diferença
entre o tempo corrente e o rótulo de tempo da tabela.
long diffTime = currentTime - timestampReal;

System.out.println(" diffTime = currentTime - timestampReal = " +
diffTime);

//Verifica se diffTime <= absoluteValidationInterval.
      if (diffTime <= absoluteValidationInterval) {

// A operação de leitura "tenta" ler o dado do banco de dados.
      opread.read();
      long imprecision = opwrite.getValue() - opread.getValue();
      |imprecision| <= allowedImprecision
       if (Math.abs(imprecision) <= allowedImprecision) {
// Imprecisão acumulada.
      long accumulatedAllowedImprecisionAux =
      accumulatedAllowedImprecision + imprecision;

      if (accumulatedAllowedImprecisionAux < 0) {
          accumulatedAllowedImprecisionAux = 0;
              }
```

```
         accumulatedAllowedImprecisionAux <=
         allowedImprecision
         if (accumulatedAllowedImprecisionAux <=
         allowedImprecision) {
        System.out.println("  CF(R,W) OK: Executada no
        prazo com Imprecisão");

//Atualiza a ImprecisaoPermitidaAcumulada
        opwrite.table.alp.accumulatedAllowedImprecision =
        accumulatedAllowedImprecisionAux;
        opread.execute();
        opwrite.write();

System.out.println("FINAL: " + opread.toString() + ", " +
opwrite.toString());
    }
// (3) accumulatedAllowedImprecisionAux > allowedImprecision
    else {
System.out.println("  CF(R,W) ERROR: Imprecisão Acumulada >
Imprecisão Permitida");
      opread.execute();
      opwrite.abort();
System.out.println("FINAL: " + opread.toString() + ", " +
opwrite.toString());
}
  }
// (2) |imprecision| > allowedImprecision
    else {
System.out.println(" CF(R,W) ERROR: Imprecisão > Imprecisão
Permitida");

      opread.execute();
      opwrite.abort();
System.out.println("FINAL: " + opread.toString() + ", " +
opwrite.toString());
    }
} Não houve compatibilidade temporal porque diffTime >
```

```
absoluteValidationInterval.

else {
System.out.println("  CF(R,W) ERROR: Dado inválido
    temporalmente");
     opread.execute();
     opwrite.reschedule();
System.out.println("FINAL: " + opread.toString() + ", " +
opwrite.toString()); }
```

A.4 Classe *MethodWriteRead*

```
public class MethodWriteRead {
    public MethodWriteRead() {
        super();
    }
    public void execute(OperationAdapter opwrite,
    OperationAdapter opread) {

        long timestampReal = opread.table.atp.timestampReal;
        long absoluteValidationInterval =
        opread.table.atp.absoluteValidationInterval;
        long allowedImprecision =
        opread.table.alp.allowedImprecision;
        long accumulatedAllowedImprecision =
        opwrite.table.alp.accumulatedAllowedImprecision;
        System.out.println("CF (W,R) Invocada");

        // VALIDAÇÃO PARÂMETROS TEMPORAIS

        long currentTime = new Date().getTime();

        long diffTime = currentTime - timestampReal;

        if (diffTime <= absoluteValidationInterval) {
```

```
    opread.read();

    long imprecision = opwrite.getValue() - opread.getValue();

    // (2) |imprecision| <= allowedImprecision
    if (Math.abs(imprecision) <= allowedImprecision) {

    // Imprecisão acumulada.
        long accumulatedAllowedImprecisionAux =
        accumulatedAllowedImprecision + imprecision;

        if (accumulatedAllowedImprecisionAux < 0) {
            accumulatedAllowedImprecisionAux = 0;
        }

   // (3) accumulatedAllowedImprecisionAux <= allowedImprecision
   if (accumulatedAllowedImprecisionAux <= allowedImprecision) {
       System.out.println("  CF(W,R) OK: Executada no prazo
       com Imprecisão");

  //Atualiza a ImprecisaoPermitidaAcumulada
      opwrite.table.alp.accumulatedAllowedImprecision =
      accumulatedAllowedImprecisionAux;
      opwrite.execute();
      opread.execute();
      opwrite.write();
      System.out.println("FINAL: " + opwrite.toString() + ", "
      + opread.toString());
  }
// (3) accumulatedAllowedImprecisionAux > allowedImprecision
   else {
  System.out.println("  CF(W,R) ERROR: Imprecisão Acumulada >
  Imprecisão Permitida");

  opwrite.execute();
```

```
          opread.abort();
          System.out.println("FINAL: " + opwrite.toString() +
          ", " + opread.toString());
      }
  }
       // (2) |imprecision| > allowedImprecision
           else {
         System.out.println("  CF(W,R) ERROR: Imprecisão >
         Imprecisão Permitida");

         opwrite.write();
         opread.abort();
         System.out.println("FINAL: " +  opwrite.toString() +
         ", " + opread.toString());
     }
 }
     // (1) Não houve compatibilidade temporal porque diffTime >
     // absoluteValidationInterval.
        else {
            System.out.println("  CF(W,R) ERROR: Dado inválido
            temporalmente");

        opwrite.write();
        opread.reschedule();
        System.out.println("FINAL: " +  opwrite.toString() + ", "
        + opread.toString());
        }
    }
}
```

A.5 Classe *MethodWriteWrite*

```
package rtdbms.protocol;

import java.util.Date;
```

```
import rtdbms.operation.OperationAdapter;

public class MethodWriteWrite {

    public MethodWriteWrite() {}

 public void execute(OperationAdapter opwrite1 /*operação executanda*/,
OperationAdapter opwrite2 /*operação invocanda*/) {

long timestampReal = opwrite2.table.atp.timestampReal;

long absoluteValidationInterval =
   opwrite2.table.atp.absoluteValidationInterval;
long allowedImprecision = opwrite2.table.alp.allowedImprecision;
long accumulatedAllowedImprecision =
  opwrite2.table.alp.accumulatedAllowedImprecision;
System.out.println("CF (W,W) Invocada");
        long currentTime = new Date().getTime();
        long diffTime = currentTime - timestampReal;

      System.out.println(" diffTime = currentTime - timestampReal = " +
diffTime);


     if (diffTime <= absoluteValidationInterval) {
     long imprecision = opwrite1.getValue() - opwrite2.getValue();

         |imprecision| <= allowedImprecision
         if (Math.abs(imprecision) <= allowedImprecision) {

             // Imprecisao acumulada.
             long accumulatedAllowedImprecisionAux =
             accumulatedAllowedImprecision + imprecision;
             if (accumulatedAllowedImprecisionAux < 0) {
                 accumulatedAllowedImprecisionAux = 0;
             }
              accumulatedAllowedImprecisionAux <=
```

```
          allowedImprecision
        if (accumulatedAllowedImprecisionAux <=
          allowedImprecision) {
            System.out.println("  CF(W,W) OK:
            Executada no prazo com Imprecisão");
               opwrite1.table.alp.accumulatedAllowedImprecision =
               accumulatedAllowedImprecisionAux;
               opwrite2.write();

               System.out.println("FINAL: " + opwrite1.toString() +
                ", " + opwrite2.toString());
            }
          accumulatedAllowedImprecisionAux >
          allowedImprecision
            else {
               System.out.println("  CF(W,W) ERROR:
               Imprecisão Acumulada > Imprecisão Permitida");
               opwrite1.execute();
               opwrite2.abort();
               System.out.println("FINAL: " + opwrite1.toString() +
                ", " + opwrite2.toString());
            }
        }
           |imprecision| > allowedImprecision
        else {

            opwrite1.write();
            System.out.println("FINAL: " +  opwrite1.toString() +
            ", " + opwrite2.toString());
        }
    }

    else {
        System.out.println("  CF(W,W) ERROR:
         Dado inválido temporalmente");
        opwrite1.write();
        opwrite2.reschedule();
```

```
            System.out.println("FINAL: " +  opwrite1.toString() +
            ", " + opwrite2.toString());
        }
    }

}
```