Disponibilidade, Detecção e Recuperação de Erros no SGBD Interbase 4.0 Utilizando Software Livre

Paulo Ricardo Rodegheri (paulor@inf.ufrgs.br)
Taisy Silva Weber (taisy@inf.ufrgs.br)

PPGC – Instituto de Informática – Universidade Federal do Rio Grande do Sul – UFRGS
Av. Bento Gonçalves, 9500 – Bairro Agronomia – 91501-970 – Porto Alegre – RS

Resumo

Este trabalho explora a aplicação de técnicas de injeção de falhas para avaliação dos mecanismos de detecção e recuperação de falhas e dos tempos necessários para disponibilizar o banco de dados aos usuários após a ocorrência de uma falha que tenha provocado um *crash* (colapso sem perda de estado). Os recursos utilizados nos experimentos são o SGBD centralizado INTERBASE versão 4.0, o sistema operacional Linux Conectiva versão 4.2 *kernel* versão 2.2.13 rodando sobre um computador Intel Pentium 550 MHz com 128 MB de memória RAM.

Abstract

This work explores the application of fault injection techniques for evaluation of the fault detection and of the recovery mechanism and of the necessary times for to available the database to the users after the occurrence of a fault that has provoked a crash (collapse without state loss). The resources used in the experiments are the centralized DBMS INTERBASE version 4.0, the operating system Linux Conectiva version 4.2 Kernel version 2.2.13 running on a computer Intel Pentium 550 MHz with 128 MB of RAM memory.

1 Introdução

Aplicações de bancos de dados são, tradicionalmente, um campo com necessidade de tolerância á falhas, principalmente visando integridade de dados e disponibilidade. Aplicações que empregam Sistemas Gerenciadores de Bancos de Dados (SGBDs) requerem que a integridade dos dados armazenados seja preservada durante a operação normal, bem como após a recuperação ocasionada por uma falha (*crash*) [1]. Falhas que afetam o hardware e os meios de armazenamento são inevitáveis. Desta forma, uma parte essencial no sistema de banco de dados é um esquema de recuperação responsável pela detecção de erros e pela restauração do banco de dados para um estado consistente, que existia antes da ocorrência da falha.

Este trabalho se propõe a avaliar os mecanismos de detecção e de recuperação de falhas, e a disponibilidade do banco de dados ao usuário após a ocorrência de uma falha do tipo *crash*, através de experimentos que provoquem, de forma deliberada, a indisponibilidade ou queda do sistema quando o SGBD estiver processando um conjunto de transações. Os experimentos provocam a indisponibilidade do banco de dados através de duas técnicas, que apesar de serem simples, obtém o resultado desejado, isto é, a queda do sistema e a interrupção dos serviços fornecidos pelo sistema de banco de dados.

2 Modelo de Falhas e Injeção de Falhas

Os experimentos no projeto enfocam falhas do tipo *crash.* Como não existe a disponibilidade do código do SGBD nesse projeto, propomos que o melhor nível de atuação para o injetor é o nível do sistema operacional, onde acessos ao meio de armazenamento e gerência dos processos podem ser facilmente manipulados alterando o código do sistema operacional ou interceptando chamadas de sistema.

A estrutura mais largamente usada para registrar modificações no banco de dados é o *log* [4]. Na ocorrência de uma falha de sistema, o *log* permite levar o banco de dados estável a um estado consistente. Como só é possível confiar no armazenamento estável, o SGBD armazena informações de *log* no armazenamento estável além do banco de dados.

3 Plataforma Experimental

O ambiente operacional é composto de softwares livres: o sistema operacional Linux e o SGBD INTERBASE 4.0. A forma de condução dos experimentos é descrita abaixo, nos capítulos 4.1 e 4.2.

3.1 O SGBD Interbase

O Interbase é um SGBD relacional, disponível gratuitamente para plataforma Linux, e possui diversas características que o diferenciam de seus concorrentes. Tradicionais SGBDs comerciais (Sybase, SQL Server, Oracle, Progress) utilizam mecanismos de *log* e de pontos de verificação para obter um estado consistente após a ocorrência e detecção de uma falha. Estes mecanismos geralmente geram uma sobrecarga de processamento e são potencialmente lentos. O Interbase não usa o conceito de *log* de transações e de pontos de verificação. Ao invés disto, mantém informações em TIPs (Transaction Information Pages) [5].

No caso de uma falha de sistema, tão logo o servidor é posto *on-line*, o Interbase automaticamente busca nas TIPs por transações ainda não concluídas (*uncommitted)*. Qualquer transação encontrada neste estado é desfeita e o sistema é imediatamente disponibilizado. Segundo o fornecedor, restaurações automáticas após uma falha de *crash* levam tipicamente menos de um segundo, e não necessitam da intervenção do administrador como na maioria dos bancos de dados [5].

4 Os Experimentos

Os experimentos foram realizados em tabelas que continham de 60.000 a 450.000 registros, sendo que cada registro lógico das tabelas ocupa 78 *bytes*. As aplicações de inserção, alteração e deleção de dados foram escritas em *SQL*, com números variáveis de registros.

4.1 Escritas no Banco de Dados

Por *default*, o Interbase executa escritas *bufferizadas*. Isto é, ele não escreve fisicamente dados no disco até que um evento pré-definido ocorra. Este evento pode ser quando uma certa quantidade de informação tenha sido coletada para uma escrita, um evento associado tenha ocorrido, ou um certo intervalo de tempo tenha passado. Se escritas forçadas (*forced writes – FW*) não estão disponíveis, igualmente o Interbase executa uma escrita interna. Os dados podem não ser fisicamente escritos em disco, porque o sistema operacional utiliza *buffers* para escrita em disco. Se ocorrer uma falha de sistema antes da escrita de dados em disco, então informações podem ser perdidas. Executando-se escritas forçadas garante-se integridade dos dados e segurança, mas perde-se desempenho [5].

4.2 Primeira Técnica

A primeira técnica consiste em matar (*kill*) um processo pai (uma sessão SQL, chamada *isql*) que contém um processo filho (um *script* SQL), que estabelece atualizações no banco de dados. No total foram executados 350 experimentos, descritos a seguir, acompanhados de seus resultados e cujo resumo é mostrado na tabela 1:

- 100 execuções de um processo que inclui 60.000 registros em uma tabela que contém 240.000 registros: 50 execuções **sem** FW, e 50 execuções **com** FW; 50 execuções de um processo que altera um campo tipo *varchar* de 50 bytes em 240.000 registros de uma tabela, **sem** FW; 100 execuções de um processo que altera um campo tipo *varchar* de 50 bytes em 300.000 registros de uma tabela, **com** FW; 100 execuções de um processo que exclui todos os registros de uma tabela que contém 300.000 registros: 50 execuções **sem** FW e 50 execuções **com** FW.

OPERAÇÃO REALIZADA	***# (KILL) PROCESSOS PAI***	***# RECUPERAÇÃO COM SUCESSO***	***% SUCESSO***	***# RECUPER. COM ERRO***	***% ERRO***
Inserção 60.000 regs- **Sem FW**	50	50	100,00%	0	0,00%
Inserção 60.000 regs- **Com FW**	50	50	100,00%	0	0,00%
Alteração 240.000 regs –**Sem FW**	50	50	100,00%	0	0,00%
Alteração 300.000 regs – **Com FW**	100	99	99,00%	1	1,00%
Deleção 300.000 regs – **Sem FW**	50	50	100,00%	0	0,00%
Deleção 300.000 regs- **Com FW**	50	50	100,00%	0	0,00%

Tabela 1: Eficiência de Recuperação Matando Processo Pai

4.3 Segunda Técnica

A segunda técnica consiste no *reset* geral do equipamento no momento em que um processo estiver sendo executado e estabelece atualizações no banco de dados, **com** e **sem** FW. No total foram executados 75 experimentos, descritos a seguir e acompanhados de seus resultados e cujo resumo é mostrado na tabela 2.

- 12 execuções de um processo que lê 60.000 registros de uma tabela e insere-os em outra tabela que contém 240.000, **sem** FW; 10 execuções de um processo que lê 60.000 registros de uma tabela e insere-os em outra tabela que contém 300.000, **com** FW; 10 execuções de um processo que lê 150.000 registros de uma tabela e insere-os em outra tabela que contém 300.000, **com** FW; 13 execuções de um processo que altera um campo tipo *varchar* de 50 bytes em 300.000 registros de uma tabela, **sem** FW; 10 execuções de um processo que altera um campo tipo *varchar* de 50 bytes em 300.000 registros de uma tabela, **com** FW; 10 execuções de um processo que deleta 450.000 registros de uma tabela, **sem** FW; 10 execuções de um processo que deleta 350.000 ou 450.000 registros de uma tabela, **com** FW.

OPERAÇÃO REALIZADA	***# RESET GERAL NO EQUIPAMENTO***	***# RECUPERAÇÃO COM SUCESSO***	***% SUCESSO***	***# RECUPER COM ERRO***	***% ERRO***
Inserção de 60.000 regs. - **sem FW**	12	9	75,00%	3(*)	25,00%
Inserção de 60.000 regs. - **com FW**	10	10	100,00%	0	0,00%
Inserção de 150.000 regs. - **com FW**	10	10	100,00%	0	0,00%
Alteração de um campo em 300.000 regs. - **sem FW**	13	7	53,80%	6(*)	46,20%
Alteração de um campo em 300.000 regs. - **com FW**	10	3	30,00%	7(*)	70,00%
Deleção de 450.000 regs. de uma tabela - **sem FW**	10	1	10,00%	9(*)	90,00%
Deleção de 350.000 ou 450.000 regs. - **com FW**	10	2	20,00%	8(*)	80,00 %

(*) = Em todos os casos houve a detecção do erro, isto é, uma mensagem foi exibida ao usuário.

Tabela 2: Eficiência de Recuperação com *Reset* Geral no Equipamento

5 Conclusões

O mecanismo de recuperação mostrou-se bastante confiável quando foi utilizada a primeira técnica, que consiste em eliminar (*kill*) um processo pai que contém um processo filho que estabelece atualizações no banco de dados. Como mostram os dados da tabela 1, o mecanismo de recuperação não foi eficiente uma única vez (0,29%), sendo, portanto eficiente em 99,71 % dos casos.

Entretanto, o mesmo mecanismo mostrou-se ineficiente quando utilizada a segunda técnica (deve-se entender o *reset* geral como um procedimento extremo e *estressante*, que no caso era executado quando um conjunto grande de transações estabelecia atualizações no banco de dados).

Nos experimentos de inserção de dados houve uma sensível diferença com o uso do parâmetro *forced writes*. Nos 20 experimentos de inserção, nos quais se utilizou o tal parâmetro, o mecanismo mostrou-se extremamente eficiente, pois em todos os casos (100%), houve sucesso no processo de recuperação. Entretanto, nos 12 experimentos de inserção em que não se utilizou o parâmetro, o mecanismo não obteve sucesso em três deles (25%). O mecanismo de recuperação, com processos de inserção, com e sem o parâmetro *forced writes,* obteve uma média geral de sucesso, de 86,36 e 90,62%.

Os experimentos de alteração de dados apresentaram resultados surpreendentes, uma vez que, esperava-se melhor desempenho do mecanismo de recuperação com o uso do parâmetro *forced writes,* visto que este é utilizado com o propósito de garantir integridade e segurança. Num total de 13 execuções, o mecanismo mostrou-se eficiente em sete, (53,8%). Nas outras seis, o mecanismo não obteve sucesso (46,2%). Dos 10 experimentos com o parâmetro *forced writes*, em apenas três o mecanismo de recuperação obteve sucesso (30%), nos demais (70%) não obteve sucesso. O mecanismo de recuperação, com os experimentos de alteração, com e sem o parâmetro *forced writes* apresentaram uma média geral de sucesso de 43,47%.

Nos experimentos de deleção de dados houve uma pequena diferença com e sem o uso do parâmetro

forced writes. Como mostram os dados da tabela 2, os experimentos com o parâmetro apresentaram melhores resultados: num total de 10 execuções, o mecanismo mostrou-se eficiente em duas (20%). Nas outras oito execuções (80%) o mecanismo não obteve sucesso. Dos 10 experimentos sem o parâmetro *forced writes*, em apenas um (10%), o mecanismo de recuperação obteve sucesso. Nas demais nove, (90%) não obteve sucesso. Fazendo-se uma média geral dos experimentos de deleção, com e sem o parâmetro *forced writes* obtém-se uma média geral do mecanismo de recuperação de 15%.

Portanto, pode-se concluir que o mecanismo de recuperação comportou-se satisfatoriamente apenas com as operações de inserção. Nas operações de alteração e deleção o mecanismo mostrou-se bastante ineficiente, mesmo quando utilizado o parâmetro *forced writes*.

Conclui-se, também, que o mecanismo de detecção de erros do Interbase é bastante eficiente, porque, de todas ás vezes, com exceção de uma (99,76 % dos casos), que o mecanismo de recuperação não conseguiu levar o banco de dados a um estado consistente após um erro, quando da tentativa de acesso ao banco de dados, uma mensagem de erro foi exibida e o acesso não foi permitido. Este tipo de comportamento é interessante e desejável, uma vez que um banco de dados que esteja em um estado inconsistente e permita acesso aos seus dados pode gerar informações inconsistentes.

O Interbase mostrou-se bastante eficiente também no quesito disponibilidade. O tempo necessário para disponibilizar o banco de dados ao usuário foi extremamente baixo (aproximadamente um segundo), mesmo quando o conjunto de operações e o tamanho das tabelas atualizadas eram grandes. Esta eficiência no desempenho é graças ao mecanismo utilizado pelo Interbase, que utiliza TIPs ao invés de arquivos de *log*.

Embora não esteja especificado como um dos objetivos deste trabalho, vale ressaltar a robustez e a confiabilidade oferecida pelo Linux. Nos 75 experimentos realizados em que foi utilizada a técnica de *reset* geral, em apenas um (1,33%) houve a necessidade do uso de *force check* (*fsck*) manual. Nos demais 74 casos, o que representa 98,67%, houve a recuperação automática e a conseqüente disponibilidade do ambiente operacional.

Referências Bibliográficas

[1] SABARATNAM, M., TORBJORNSEN, O., Cost of Ensuring Safety in Distributed Database Management Systems**.** *Proceedings of the 1999 Pacific Rim International Symposium on Dependable Computing.*

[2] COSTA, D., MADEIRA, H., Experimental Assessment of COTS DBMS Robustness under Transiente Faults**.** (http://dsg.dei.uc.pt) – Jul/00.

[3] KORTH H. F., SILBERSCHATZ, A., Sistema de Banco de Dados – Segunda edição, São Paulo: Makron Books, 1995.

[4] BERNSTEIN, P., HADZILACOS, V., GOODMAN, N. Concurrency Control and Recovery in Database Systems. Reading, Massachusetts : Addison-Wesley, , 1987.

[5] A Comparasion of Interbase vs. SQL Server**.** (http://www.borland.com/interbase/papers) – Jul/00.