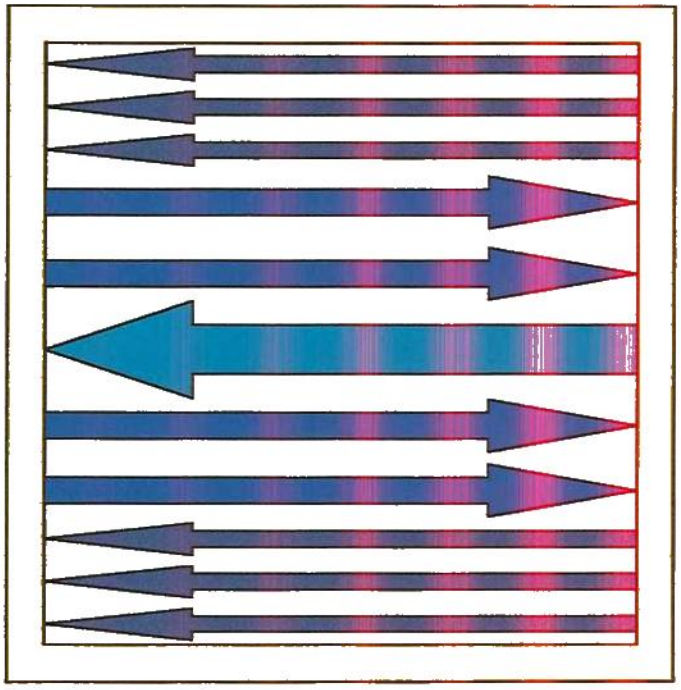
I Workshop de Tolerância a Falhas

ANAIS



ANAIS

I WTF

14 e 15 de Maio de 1998
Porto Alegre - RS

Elias Procópio Duarte Júnior

Universidade Federal do Rio Grande do Sul
Instituto de Informática

I Workshop de Tolerância a Falhas

Porto Alegre – RS
14 e 15 de maio de 1998

ANAIS

Editado por
Mário Magalhães Leboute

Realização
Instituto de Informática
Universidade Federal do Rio Grande do Sul

Promoção
Instituto de Informática – UFRGS

Apoio
FAPERGS
PROPESQ-UFRGS
SBC-RS

Capa: Mário Magalhães Leboute

Dados para Catalogação na Publicação (CIP)

I Workshop de Tolerância a Falhas (1.: 1998, mai. 14-15: Porto Alegre)
Anais / editado por Mário Magalhães Leboute. – Porto Alegre : Instituto de Informática da UFRGS, 1998.
108 p.

1. Confiabilidade de Computadores. 2.Tolerância a Falhas. I. Leboute, Mário Magalhães. II. I WTF (1.:1998). III. Título.

Cópias adicionais:

Universidade Federal do Rio Grande do Sul
Instituto de Informática / Setor de Eventos
Caixa Postal 15064
91501-970 – Porto Alegre – RS
e-mail: eventos@inf.ufrgs.br

Apresentação

A área de tolerância a falhas tem importância cada vez mais decisiva em computação. A dependência cada vez maior dos sistemas informatizados torna indispensável a adição de confiabilidade como um recurso básico para a utilização de inúmeros sistemas.

A idéia da realização do I WTF surgiu em dezembro de 1996 com o I Simpósio Regional de Tolerância a Falhas, financiado pela Fapergs. Este primeiro simpósio foi uma experiência muito rica de comunicação entre pesquisadores de Tolerância a Falhas no estado do Rio Grande do Sul, e serviu para consolidar o contato entre grupos de pesquisa. Entre os progressos na área de tolerância a falhas no estado, se verificaram um aumento nas publicações nacionais e internacionais, o surgimento de novas linhas de pesquisa, a formação de mestres e o início de projetos de cooperação universidade-empresa para o desenvolvimento de sistemas confiáveis.

O I WTF busca a extensão e consolidação destes avanços, ao congregar pesquisadores, professores, profissionais e estudantes de computação das regiões Sul e Sudeste do país para a troca de experiências. Visa também fazer um levantamento dos trabalhos na área de tolerância a falhas de pesquisadores do RS e estados vizinhos, que desenvolvem trabalhos complementares e/ou que possam ser incorporados a trabalhos de outras instituições. Esse levantamento, e a oportunidade de discutir com colegas, permitirá qualificar os trabalhos de instituições emergentes e enriquecer a pesquisa nas instituições consolidadas.

O I WTF é organizado pelo Instituto de Informática da Universidade Federal do Rio Grande do Sul. Conta com um comitê de programa formado por professores e pesquisadores da UFRGS, Unicamp, PUCRS, UCS, UniCruz, UFSM, UFSC e Unisc, instituições onde as atividades de ensino e pesquisa na área de tolerância a falhas se encontram em pleno desenvolvimento.

I WTF

I Workshop de Tolerância a Falhas

Coordenação Geral:
Taisy Silva Weber (UFRGS)

Coordenação Editorial:
Mário Magalhães Leboute (UFRGS)

Comitê de programa:
Cecília Mary Fischer Rubira (UNICAMP)
Eduardo Augusto Bezerra (PUCRS)
Eliane Martins (UNICAMP)
Ingrid E. S. Jansch Porto (UFRGS)
Luciano Porto Barreto (UCS)
Marco Aurélio Spohn (UNICRUZ)
Maria Lúcia Blanck Lisbôa (UFRGS) – presidente
Patrícia Pitthan Barcelos (PUCRS)
Raul Ceretta Nunes (UFSM)
Raul Fernando Weber (UFRGS)
Rômulo Silva de Oliveira (UFRGS)
Taisy Silva Weber (UFRGS)
Werner Haetinger (UNISC)

Comitê Organizador:
Alessandro Agnoletto (UFRGS)
André Peres (UFRGS)
Fernanda Kruel Denardin (UFRGS)
Mário Magalhães Leboute (UFRGS)

Sumário

SESSÃO 6: IMPLEMENTAÇÃO DE SERVIÇOS CONFIÁVEIS

SESSÃO 7: TOLERÂNCIA A FALHAS E LINGUAGENS DE PROGRAMAÇÃO - II

Uma Experiência com o Modelo de Programação Orientada a Grupos

Jeferson Botelho do Amaral[♠] Raul Ceretta Nunes[♦]

Departamento de Eletrônica e Computação
CT – UFSM – Campus Camobi – Santa Maria – RS – CEP 97105-900
Tel. (055) 220 8418 – Fax: (055) 220 8030 – e-mail: *delc@ct.ufsm.br*

Resumo

A coordenação segura e consistente da cooperação entre processos num sistema distribuído sob falhas exige muita habilidade do programador. Primitivas para comunicação confiável entre grupo de processos criam uma estrutura básica poderosa para o desenvolvimento de aplicações distribuídas.

Este artigo descreve uma experiência de programação usando o sistema *x*AMp (*Extended Atomic Multicast Protocol*), um serviço de comunicação de grupo altamente versátil. Ele consiste de um pacote integrado, projetado para ser usado sobre redes locais. O objetivo é o desenvolvimento de uma aplicação simples, neste caso replicação de dados, usando o paradigma de grupo de processos.

Palavras-chave: comunicação de grupo, *x*AMp, replicação de dados

Abstract

In distributed systems, to reach secure and consistent processes coordenation demand expert programmers. Reliable group communication primitives create a powerful framework for the development of distributed applications.

This paper describes a programming experience using the *x*AMp system (*Extended Atomic Multicast Protocol*), a highly versatile group communication service. It consists of an integrated package, designed to be used over local-area networks. The goal is the development of a simple application, in this case data replication, using the process group paradigm.

Keywords: group communication, *x*AMp, data replication

1 Introdução

Desenvolver aplicações distribuídas robustas exige do programador um grande esforço para driblar os problemas oriundos de defeitos típicos na comunicação, principalmente quando é desejado que as aplicações possuam tolerância a falhas. No desenvolvimento destas aplicações, as tarefas e a cooperação entre os processos apontam para uma modelagem baseada no conceito de grupo de processos [BIR96]. A existência de primitivas para comunicação confiável entre os processos pertencentes a um grupo são vitais na determinação da confiabilidade e conseqüente possibilidade de suporte a falhas da aplicação. Sistemas com tolerância a falhas têm usado primitivas de comunicação de grupo para propagar atualizações em réplicas e para fazer, por exemplo, decisões distribuídas ao colecionar quorum num sistema de votação.

[♠] Bacharel em Informática pela UFSM – e-mail: *amaral@inf.ufsm.br*
[♦] Prof Msc. do DELC/CT – UFSM – e-mail: *ceretta@inf.ufsm.br*

Padronizações para o modelo de comunicação de grupo estão em estudo no X/Open e no IEEE, e um padrão de comunicação paralela orientada a grupo *Message-Passing Interface* [MPI97] foi introduzido em 1993. Alguns sistemas operacionais distribuídos têm fornecido mecanismos para a comunicação de grupo, adicionalmente aos métodos mais convencionais como o RPC (*Remote Procedure Call*) e o mecanismo de *stream* [BIR96]. Entretanto, com poucas exceções como o Amoeba [TAN91], fornecem muitas limitações no suporte a características como confiabilidade e consistência, que são essenciais em aplicações que executam durante um tempo longo ou em aplicações críticas. A desvantagem do Amoeba está na pouca portabilidade para outros sistemas proprietários.

Há uma variedade de ferramentas fornecendo suporte à programação de aplicações que desejam usar comunicação de grupo confiável. **Isis** [BIR87] foi o primeiro sistema a introduzir primitivas de comunicação de grupo; foi desenvolvido inicialmente como um projeto acadêmico na Universidade de Cornell e mais tarde tornou-se um produto comercial. Desde o Isis, muitas outras ferramentas têm surgido: **Transis** [AMI92], **Phoenix** [MAL95], **Relacs** [BAB95], ***x*AMp** [ROD92], **Horus** [REN95], etc. Estas ferramentas diferem nas primitivas de *multicast* que oferecem e, também, na forma de interfaceamento com as subcamadas do sistema distribuído.

Este trabalho descreve uma experiência prática utilizando o modelo de programação orientada a grupos. A aplicação escolhida e modelada foi **replicação de arquivos**, usando a técnica de cópia primária [JAL94]. O ambiente de programação utilizado foi o sistema operacional Linux, com primitivas de comunicação de grupo confiável fornecidas pela ferramenta *x*AMp. Como o objetivo foi apenas experimental, não se levou em conta todos os níveis de complexidades inerentes a este tipo de aplicação.

2 Comunicação de Grupo

A propriedade fundamental de grupo de processos é que quando uma mensagem é enviada para o grupo, todos os membros devem recebê-la, uma forma de comunicação **um-para-muitos** (***multicast***). Este tipo de comunicação contrasta com a comunicação **ponto-a-ponto** (***unicast***), onde um processo só é capaz de enviar uma mensagem para um outro processo.

Na utilização da comunicação de grupo, notam-se alguns aspectos semelhantes ao modelo comum de troca de mensagens [TAN92], por exemplo: **bufferização** e **não-bufferização**; **bloqueio** e **não-bloqueio**. Entretanto, há muitos detalhes novos, pois o envio de mensagens para um grupo de processos difere do envio a um único processo, e a estrutura interna dos grupos pode ser organizada de diversas formas.

Em **grupos fechados**, somente os membros podem enviar mensagens para o grupo; processos que não pertençam ao grupo não podem enviar mensagens para o grupo como um todo, embora possam fazê-lo para membros individuais do grupo. Por outro lado, na estruturação através de **grupos abertos** não há esta propriedade, pois qualquer processo do sistema é capaz de enviar mensagens para qualquer grupo.

Em alguns grupos há uma igualdade entre os processos, nenhum é superior, e todas as decisões são tomadas coletivamente (**grupos não-hierárquicos**). Em outros grupos, os processos são organizados hierarquicamente (**grupos hierárquicos**), podendo, neste caso, existir um processo coordenador com missão de gerenciar as tarefas dos demais processos no grupo.

Em **grupos estáticos**, os membros não podem deixar o grupo e nem novos processos podem juntar-se a ele. Já em **grupos dinâmicos**, novos grupos podem ser criados ao passo que grupos antigos podem ser destruídos, e um processo pode juntar-se ao grupo ou sair do grupo. Adicionalmente, um processo pode ser membro de vários grupos ao mesmo tempo, característica esta denominada **sobreposição**.

3 *x*AMp

O *x*AMp (*Extended Atomic Multicast Protocol*) [ROD92], desenvolvido no Instituto de Engenharia de Sistemas de Computadores da Universidade de Lisboa, é uma ferramenta de suporte ao desenvolvimento de aplicações distribuídas com exigências de performance, funcionalidade e dependabilidade. Fornece serviços para o gerenciamento de membros do grupo, permitindo a criação dinâmica e a reconfiguração de grupos de processos. Também permite que durante o tempo de vida de um grupo, processos possam juntar-se a um grupo (*join*) ou deixá-lo (*leave*). A falha de um membro do grupo é detectada e uma indicação do evento é enviada aos membros remanescentes do grupo.

A ferramenta *x*AMp fornece um suporte eficiente e versátil para a troca de informações entre os membros do grupo, um serviço de comunicação *multicast*. O serviço aceita uma lista de endereços, chamado **endereço seletivo**, como um endereço de destino válido para uma mensagem *multicast*, assim pode enviar, transparentemente, uma mensagem para os destinatários. Um **endereço lógico**, adicionalmente, pode ser associado com um grupo, permitindo a todos os membros do grupo serem endereçados através de um **nome lógico**. Isto libera o programador de ter que, explicitamente, gerenciar listas de endereços seletivos.

Outra característica do *x*AMp, é fornecer um ambiente de execução com propriedades de **validade** e **sincronismo**, as quais são relevantes para a maioria dos sistemas de comunicação. Isto faz com que o usuário possa confiar no sistema, no sentido de que as mensagens não serão corrompidas, não serão perdidas arbitrariamente, nem mesmo geradas espontaneamente.

Propriedades de **acordo** surgem quando do envio de mensagens *multicast*; neste conjunto, a propriedade de **unanimidade** é a que fornece maiores garantias, pois se uma mensagem for enviada a um membro correto, será enviada a todos os membros corretos, mesmo na presença de falhas. Entretanto, este tipo de propriedade, pode não ser tão interessante para alguns tipos de aplicações, pois degrada o desempenho. Por exemplo, pedidos a servidores replicados precisam alcançar apenas uma das réplicas, ao invés de todas, uma vez presumido que todas as respostas são idênticas. Protocolos baseados em quorum são outro exemplo onde a propriedade de unanimidade não é exigida. Devido a fatos deste tipo, o *x*AMp possui diversas propriedades de acordo.

Finalmente, o *x*AMp possui propriedades que especificam o tipo de **ordenação** que o protocolo deve utilizar para a troca de mensagens entre seus membros. O serviço de comunicação de grupo do *x*AMp proporciona diversas primitivas, intencionando satisfazer a maioria das exigências das aplicações com respeito à comunicação de grupo. A **ordenação total** é a propriedade que fornece maiores garantias, pois as mensagens são enviadas na mesma ordem para os diferentes membros. **Ordenação causal** e **FIFO** provêm menores garantias, contudo há uma redução no custo, o que melhora o desempenho em aplicações que não exigem ordenação total.

4 Replicação de Arquivos e a Técnica de Cópia Primária

Replicação de arquivos se insere no contexto de várias tecnologias criadas para se conseguir maior dependabilidade nos sistemas de computação. A replicação pode ser conseguida através de técnicas com enfoque no *hardware* ou em mecanismos de *software* (utilizados neste trabalho).

As técnicas de replicação por *software* baseiam-se em sistemas com processadores do tipo *fail-stop* e enquadram-se em três classes: **técnica de cópia primária**, **técnica de cópias ativas** e **técnica de quorum**. O modelo de grupo de processos adequa-se bem tanto à técnica de cópias ativas quanto a de cópia primária. Em decorrência do caráter experimental deste trabalho, foi escolhida a técnica de cópia primária, pela maior simplicidade de implementação. Esta técnica garante que operações sobre os dados possam ser realizadas, mesmo se alguns nós que estejam atuando como servidores de arquivos, ou as comunicações, falhem [JAL94].

Na especificação original da técnica de cópia primária, para que seja possível suportar a falha de n nós, é necessária a existência de, no mínimo, $n+1$ nós. Um destes nós é denominado como *primário*, e a cópia de um volume existente nele é conhecida como *cópia primária*. O restante dos nós são denominados *backups*. Um servidor *backup* não interage diretamente com os clientes, mas somente com o servidor primário. Portanto, há um centralizador de pedidos na técnica, o que facilita a ordenação dos pedidos. O servidor primário ao receber um pedido de leitura interage somente com o leitor. Mas, ao receber um pedido de escrita, deverá difundir a atualização para os seus servidores *backup* e, somente após receber a confirmação, volta a interagir com o cliente escritor.

Em caso de falha, um fator importante para manter a consistência das réplicas são as propriedades de ordenação e atomicidade aplicadas sobre as mensagens usadas na atualização das réplicas. Por exemplo, quando o servidor primário falha após já ter enviado uma mensagem de operação para qualquer um dos *backups*, mas ainda não tenha enviado a mensagem para todos os *backups*, a falha poderia fazer com que um servidor *backup* se tornasse primário sem possuir o estado mais atual da cópia. Isto pode ser evitado com o uso de um protocolo de difusão de escritas atômico com comunicação confiável, similar ao fornecido pela ferramenta *x*AMp.

5 Modelagem da Aplicação

Segundo Tanenbaum [TAN92], grupos fechados são adequados ao processamento paralelo e grupos abertos, por outro lado, são adequados para aplicações do tipo cliente-servidor. Entretanto, verificou-se que as ferramentas de comunicação de grupo existentes não costumam oferecer grupos abertos. Normalmente isto não é oferecido devido à complexidade de gerenciar e implementar, de forma confiável, tal tipo de grupo. No caso do *x*AMp, ferramenta utilizada, os grupos são fechados.

A modelagem da aplicação, como mostra a figura 1, fez uso de grupos fechados (único oferecido) e dinâmicos, dado que um grupo deve definir um novo servidor primário sempre que o corrente falhar. A interação cliente-servidor foi obtida fazendo uso de 3 grupos distintos: **grupo dos leitores** (formado por processos clientes que desejam ler um arquivo), **grupo dos escritores** (formado por processos clientes que desejam alterar o conteúdo de determinado arquivo) e **grupo dos servidores** (formado pelo processo com função de servidor primário e os processos com função de *backup*).

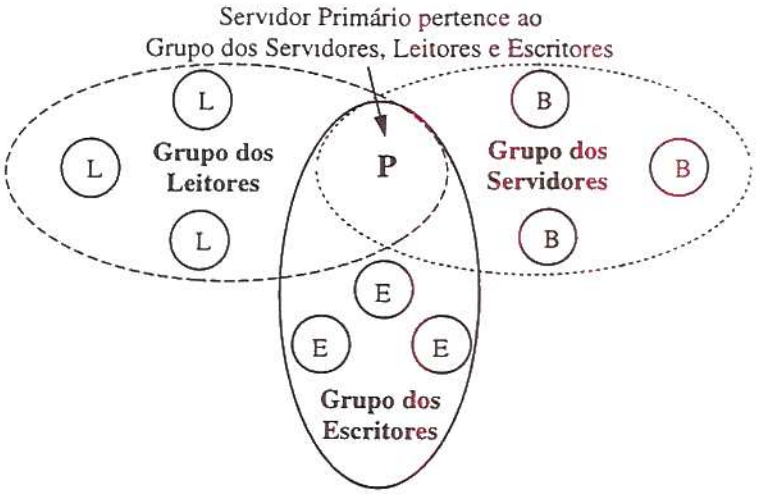


Figura 1: Modelagem utilizada na aplicação

Os processos do grupo dos servidores possuem o mesmo código, facilitando o mecanismo de eleição de um novo servidor primário que, como mostra a figura 1, apresenta característica de sobreposição. Desta forma, já que um cliente (leitor ou escritor), pertence ao mesmo grupo do servidor primário, torna-se possível a comunicação cliente-servidor. No grupo dos servidores há uma hierarquia entre o servidor primário e os servidores *backup*, pois o primário é quem os coordena. Por outro lado, pode-se dizer que o grupo também possui aspectos não-hierárquicos, pois os processos têm o mesmo código e, conseqüentemente, são capazes de desempenhar a mesma função, ou seja, um *backup* é capaz de tornar-se primário.

Para eleger um novo servidor primário, é utilizado o conceito de **visões** (em síntese, uma representação de quem é membro do grupo num determinado instante). Uma visão sofre modificação sempre que um novo membro se junta ao grupo, ou que um membro sai do grupo, de modo voluntário ou involuntário (uma falha). Cada grupo de processos possui uma lista contendo os processos membros do grupo. O conceito de visões pode ser aliado à identificação de um índice fixo na lista de membros como sendo o processo servidor primário [GUE97] [JAL94]. O *x*AMp mantém o controle da visão de cada grupo; caso um membro falhe, todos os demais são notificados da falha. Entretanto, é preciso construir um algoritmo que identifique se o membro que falhou foi o servidor primário e, em caso positivo, que inicialize um algoritmo de eleição.

6 Conclusões

Embora existam ferramentas fornecendo suporte à programação de aplicações utilizando o modelo de comunicação de grupos, o mercado ainda apresenta um número reduzido de aplicações fazendo uso desta técnica. Ao que parece, isto se deve ao pouco entendimento, por parte dos programadores, sobre como modelar diferentes classes de aplicações através de grupo de processos cooperantes.

Essa foi a principal observação deste experimento, pois a modelagem do exemplo tratado inicialmente parecia complicada, no que se refere a interação cliente-grupo (o *x*AMp usa um modelo de grupos fechados), contudo a característica de sobreposição de grupos aliada ao conceito de mudança dinâmica de visões, acabou facilitando bastante a programação Com base nesta constatação, verifica-se que a afirmação de Tanenbaum [TAN92], mencionada anteriormente na seção 5, pode ser mais uma questão de paradigma no uso de grupos do que propriamente uma adequação dos grupos abertos e fechados.

Atingiu-se o objetivo de verificação experimental do modelo de grupos de processos e a ferramenta *x*AMp mostrou-se bastante útil neste contexto, fornecendo ao programador a abstração de grupos de processos aliada a uma variedade de primitivas com características de atomicidade e ordenação. Observou-se que dispor deste tipo de ferramenta facilita bastante a programação.

Para trabalhos futuros, sugere-se implementar a mesma aplicação através de outra ferramenta que forneça a abstração de grupos, com a finalidade de realizar um comparativo levantando questões como flexibilidade, portabilidade e facilidade de programação.

7 Referências Bibliográficas

[AMI92] AMIR, Y.; DOLEV, D.; KRAMER, S.; MALKI, D. **Transis: A Communication Sub-System for High Availability**. 22nd International Symposium on Fault-Tolerant Computing (FTCS-22nd), Boston, July 1992.

[BAB95] BABAOGLU, O.; DAVOLI, R.; GIACHINI, L.A.; BAKER, M. **Relacs: A Communications Infraestruture for Constructing Reliable Apllications in Large-Scale Distributed Systems**, Technical Report UBLCS-94-15, June 1994.

[BIR87] BIRMAN, K.P.; JOSEPH, T.A. **Reliable Communication in the Presence of Failures**. ACM Trans. on Computer Systems, Vol. 5, No. 1, February 1987.

[BIR96] BIRMAN, K.P. **Building Secure and Reliable Network Applications**. Greenwich: Manning, 1996.

[GUE97] GUERRAOUI, R.; SCHIPER, A. **Software-Based Replication for Fault Tolerance**. IEEE Computer Magazine, Vol. 30, No. 4, pp. 68-74, Abril de 1997.

[JAL94] JALOTE, P. **Fault Tolerance in Distributed Systems**. New Jersey: Prentice-Hall, 1994.

[MAL95] MALLOTH, C.; FELBER, P.; SCHIPER, A.; WILHELM, U. **Phoenix: A Toolkit for Building Fault-Tolerant, Distributed Applications in Large Scale**, Départment d'Informatique, Ecole Polytechique Fédérale de Lausanne, Switzerland, July, 1995.

[MPI97] PACHECO, P.S. **Parallel Programming With Mpi**, San Francisco, Morgan Kaufmann, 1997. 418p.

[REN95] RENESSE, R.V.; BIRMAN, K. **Protocol composition in Horus**, Technical Report 95-1505, Cornell University, Dept. of Computer Science, March 1995.

[ROD92] RODRIGUES, L.; VERÍSSIMO, P. ***x*AMp: A Protocol Suite for Group Communication**, INESC, Technical University of Lisboa, January 1992.

[TAN91] TANENBAUM, A.S.; KAASHOEK, F.M.; RENESSE, R.V. **The Amoeba Distributed Operating System - A Status Report**. Department of Mathematics and Computer Science. Vrije Universiteit. Amsterdan, The Netherlands, 1991.

[TAN92] TANENBAUM, A.S. **Modern Operating Systems**. Englewwod Clifs, NJ.: Prentice-Hall, 1992. 728p.

Suporte a Tolerância a Falhas no Ambiente de Programação DPC++[1]

Maurício Lima Pilla[2] **Marcos Ennes Barreto**[3]
Rafael R. dos Santos[4] **Gerson G. H. Cavalheiro**[5]
Philippe O. A. Navaux[6]

Instituto de Informática
Universidade Federal do Rio Grande do Sul
Caixa Postal 15064, Porto Alegre, RS

Resumo

Este trabalho apresenta o suporte a Tolerância a Falhas para o ambiente de programação DPC++. O mecanismo que está sendo implementado atualmente é constituído por um algoritmo de criação e recuperação de *checkpoints*, o qual permite a um programa realizar recuperação automática de falhas de um objeto distribuído, aumentando a confiabilidade da aplicação. As aplicações DPC++ são geradas através do pré-compilador DPC++, o qual possibilita que aplicações utilizem o mecanismo de tolerância a falhas de modo transparente ao usuário.

Abstract

This paper presents Fault Tolerance support for DPC++ programming environment. The mechanism, that has been implemented, is constituted by an algorithm of creation and recovery of checkpoints, which allows a program to recover automatically from distributed object faults, increasing the reliability of applications. The DPC++ programs are generated by the precompiler, making possible the use of the fault tolerance mechanism in a transparent way to the user.

[1] Projeto financiado pelo CNPq e FINEP através do Programa PAD
[2] Bolsista IC-CNPq, bacharelando, Instituto de Informática, UFRGS, email:pilla@inf.ufrgs.br
[3] Mestrando, CPGCC, UFRGS, email:barreto@inf.ufrgs.br
[4] Doutorando, CPGCC, UFRGS, email rrsantos@inf.ufrgs.br
[5] Doutorando INPG-IMAG-França, email: Gerson.Cavalheiro@imag.fr
[6] Professor Doutor (Grenoble, 1979), Instituto de Informática, UFRGS, email.navaux@inf.ufrgs.br

1 Introdução

Em aplicações científicas e comerciais, a computação de um programa que foi interrompido por uma falha tem de ser feita desde o seu início novamente. Como resultado, as aplicações são terminadas apenas se o tempo entre falhas é menor que o tempo de execução do programa [ZOM96]. Foi demonstrado em outros estudos que o tempo médio de execução de um programa na presença de falhas cresce exponencialmente com o tamanho do programa.

Observando-se esta realidade, pode-se notar que a inclusão de um mecanismo de tolerância a falhas que permita a recuperação da computação já realizada, em caso de falha, é desejável em sistemas distribuídos. O processamento distribuído, com nodos possuindo memória e processadores próprios, interligados por uma rede de conexão, gera situações onde há redundância e esta pode ser utilizada para aumentar a confiabilidade dos sistemas distribuídos. Por exemplo, os processos que estejam executando em um nodo que falhe podem ser recuperados em outro nodo, desde que haja informação suficiente para tanto.

Uma abordagem possível é o uso de técnicas de *recuperação de erros por retrocesso* [SIN94]. *Checkpoints* (ou pontos de recuperação) são obtidos através da gravação do estado de processos em meio estável, normalmente em disco rígido. Do estado de um processo constam o valor das variáveis, seu ambiente, informações de controle, valor dos registradores, pilha, etc. Com a utilização de *checkpoints*, o tempo médio de execução de um programa passa a crescer linearmente com o tamanho do programa [ZOM96].

Como os processos interagem em ambientes distribuídos através de troca de mensagens, e a rede de comunicação possui um atraso considerável em relação a acessos à memória, o estado dos canais de comunicação passa a ser um importante fator a ser considerado na introdução de mecanismos de tolerância a falhas em sistemas distribuídos. A criação de um *checkpoint* global implica em haver uma sincronização global de processos, de modo que este *checkpoint* reflita um estado consistente em um determinado instante de tempo. Esta sincronização dos nodos componentes do sistema distribuído é extremamente complexa e cara. Uma solução possível, descrita em Singhal [SIN94], seria o estabelecimento de *checkpoints* assíncronos, não considerando a troca de mensagens entre processos, porém este método não garante que o estado das comunicações seja refletido corretamente em caso de *rollback* (recuperação).

Outro método para a criação de *checkpoints* consiste em manter um conjunto consistente de *checkpoints* [SIN94]. Cada processo tomaria seu *checkpoint* após cada envio de mensagem. Desta forma, o conjunto dos *checkpoints* mais recentes está sempre consistente. Porém, este método pode resultar em mensagens órfãs e, conseqüentemente, em um estado inconsistente do sistema. Além disto, este esquema necessita que a operação de envio ou recebimento de uma mensagem e a criação de *checkpoints* constituam uma operação atômica, algo difícil de ser implementado.

Portanto, fica clara a necessidade de um método de criação de *checkpoints* distribuídos que garanta a consistência do estado global do sistema, gerando o menor *overhead* possível para tanto.

O presente artigo aborda na seção 2 *a linguagem de programação DPC++ e o modelo distribuído*. A seção 3 consiste do *modelo de checkpoints* desenvolvido para o DPC++. A seção 4 apresenta uma *conclusão* sobre o artigo.

2 A Linguagem DPC++ e o Modelo Distribuído

Processamento Distribuído em C++ (DPC++) é uma linguagem para programação distribuída orientada a objetos baseada em C++. É uma linguagem de propósito geral, que dispõe de recursos que visam facilitar a programação de grandes sistemas distribuídos.

As características de orientação a objetos de DPC++ são as herdadas do C++. Inclusive a sintaxe dos programas é a mesma. Porém, em DPC++, foi introduzido um novo tipo de classe: a classe dos objetos distribuídos.

A simplicidade da escrita de aplicações distribuídas é provida pelo pré-processador DPC++. O usuário não precisa preocupar-se com detalhes de comunicação entre objetos distribuídos, pois o pré-processador encarrega-se de gerar o código que será submetido ao compilador C++.

Os conceitos básicos de orientação a objetos são aplicados ao modelo DPC++, onde existem objetos que encapsulam todas as suas propriedades: dados e métodos. A execução de programas é realizada através da invocação de métodos dos objetos, através do envio de mensagens. Ainda, no modelo DPC++ é explorada a concorrência de execução entre objetos, utilizando-os em sistemas distribuídos.

A linguagem DPC++ utiliza como modelo base de objetos distribuídos a execução da função destes em uma rede de processadores homogêneos, onde cada nodo pode suportar n objetos distribuídos executando, sendo este número limitado apenas pela capacidade de memória localmente disponível. Um escalonador é responsável por compartilhar o uso do processador entre os objetos.

O *Diretório* é um objeto distribuído especializado que possui tabelas com informações sobre todos os demais objetos distribuídos, tais como nodo em que está alocado, endereço de comunicação, identificador do processo e tipo do objeto. O Diretório provê serviços ligados à manutenção, como criação de novos objetos distribuídos, pesquisa sobre informações destes objetos e verificação de falha e restauração de objetos distribuídos.

Em Cavalheiro [CAV93], são encontradas outras considerações e restrições da linguagem DPC++.

3 Modelo de Checkpoints

O modelo de criação e recuperação de *checkpoints* desenvolvido para o ambiente de programação DPC++ é baseado na criação de *checkpoints* após cada troca de mensagens entre objetos distribuídos, com um protocolo para garantir que os *checkpoints* foram criados e que não há perda de mensagens e em um mecanismo de *timeout*, o qual detecta a ocorrência de erros.

Considera-se que o meio de transmissão é confiável o suficiente para garantir que todas as mensagens enviadas pelo nodo origem são corretamente recebidas pelo nodo destino [MID90]. O problema de confiabilidade do meio de transmissão não é considerado inicialmente. Falhas do objeto Diretório também não são consideradas neste momento, sendo que um outro mecanismo de tolerância a falhas está sendo desenvolvido para estes casos.

A criação de *checkpoints* foi implementada através de uma versão modificada da *libckpt* [PLA95], na qual foi adicionada a capacidade de criação de múltiplos *checkpoints* para um mesmo usuário, através do acréscimo de uma extensão que diferencia os mesmos.

O algoritmo responsável pela consistência do estado global resultante da criação dos *checkpoints* e pela eventual recuperação de objetos distribuídos que falham está descrito em Pilla [PIL97].

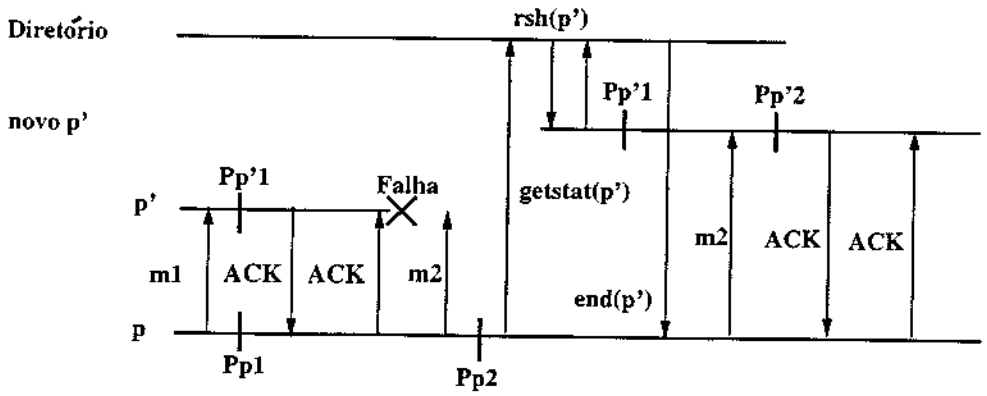


Figura 1 Fluxo de mensagens entre dois objetos distribuídos

A figura 1 ilustra o fluxo de mensagens entre dois objetos, p e p'. Inicialmente, o objeto distribuído p envia uma mensagem *m1* ao objeto distribuído p' e ambos criam seus *checkpoints*. Depois, o objeto p' envia uma mensagem de confirmação (ACK) ao objeto p, que então envia uma mensagem de confirmação para p' e prossegue normalmente.

Após enviar a mensagem de confirmação para p, p' falha. Quando p tenta enviar uma mensagem *m2* para p', cria seu *checkpoint* e aguarda pela confirmação de p'. Como p' não responde em um determinado *timeout*, p envia uma mensagem ao Diretório, pedindo o *status* do objeto p'. O Diretório descobre que p' falhou e dispara um novo objeto da mesma classe (*rsh(p')*). Este novo objeto utiliza as informações armazenadas no último estado armazenado do objeto que falhou para restaurar seu contexto e continuar a execução.

O Diretório devolve para p o novo endereço de p'. O objeto p envia novamente a mensagem, sem, contudo, criar um novo *checkpoint*, e espera pela confirmação. p' envia a confirmação e espera pela confirmação de p. Depois disto, ambos os objetos prosseguem normalmente.

O algoritmo de criação de *checkpoints* apresentado por Koo [KOO87] exige um número maior de mensagens, pois quando um processo deseja criar um *checkpoint* envia mensagens a todos os processos de quem recebeu mensagens desde a criação de seu último *checkpoint*. Estes processos, por sua vez, executam o mesmo procedimento, fazendo com que o número de mensagens seja bastante grande em função do número de processos (objetos). O outro problema associado à criação de *checkpoints* deste modo é que um grande número de processos criam *checkpoints* ao mesmo tempo, o que pode criar uma carga muito elevada para o sistema em certos momentos.

Atualmente, estão sendo desenvolvidas as bibliotecas que permitirão a incoporação do mecanismo de tolerância a falhas proposto ao pré-processador DPC++, através de um segundo protótipo do mecanismo. Com isto, será possível desenvolver aplicações distribuídas em DPC++ incluindo de forma transparente ao programador o mecanismo de tolerância a falhas.

4 Conclusão

O mecanismo apresentado permitirá que programadores possuam uma ferramenta de fácil programação com uma confiabilidade maior, decorrente do uso do mecanismo de tolerância a falhas. Como foi mostrado em Pilla [PIL97], o mecanismo suporta falhas de um objeto distribuído por vez.

Uma característica interessante do modelo adotado é que objetos que não tenham falhado não necessitam serem recuperados para um estado anterior, o que poupa o sistema deste *overhead*. Esta característica é alcançada através da utilização das particularidades dos objetos distribuídos DPC++, os quais somente se comunicam com outros objetos nos momentos de invocação de métodos e de resposta dos mesmos. Outros modelos de criação de *checkpoints* não garantem que apenas o processo que falha necessite fazer o *rollback*, sendo que em alguns casos pode ocorrer o chamado *efeito dominó*. No *efeito dominó*, um processo que falha força outro processo a voltar ao seu último *checkpoint*, devido ao estado inconsistente das comunicações. Este processo, por sua vez, faz com que outro objeto retorne ao seu último *checkpoint*, e assim sucessivamente.

Outra característica importante é a inexistência de um coordenador centralizado para a criação dos *checkpoints*, sendo que cada par de objetos distribuídos que se comunicam criam seus *checkpoints* sem afetar os demais objetos.

A implementação deste mecanismo será complementada com outros mecanismos de tolerância a falhas, tais como um mecanismo para o objeto Diretório.

Referências

[BEL93] BELMONTE, Valdir Rossi, WEBER, Raul Fernando. *Gerindo Tolerância a Falhas em Sistemas Distribuídos.* São José dos Campos: V Simpósio de Computadores Tolerantes a Falhas, *anais...*, outubro, 1993.

[CAV93] CAVALHEIRO, Gerson G. H., NAVAUX, P. O. A.. *DPC++: Uma Linguagem para Processamento Distribuído.* Florianópolis: V SBAC-PAD, *anais...*, outubro, 1993.

[CAV94] CAVALHEIRO, Gerson G. H., SANTOS, Rafael R., NAVAUX, Philippe O. A.. *Análise de Desempenho de um Protótipo da Linguagem DPC++.* Caxambu : XXI SEMISH, *anais...*, 1994.

[JAL94] JALOTE, Pankaj. *Fault Tolerance in Distributed Systems.* P T R Pretince Hall. 1994.

[KAN97] KANELLAKIS, Paris C.; SHVARTSMAN, Alex A. *Fault-Tolerant Parallel Computation.* Kluwer Academic Publishers. 1997.

[KOO87] KOO, Richard, TOUEG, Sam. *Checkpointing and Rollback-Recovery for Distributed Systems.* IEEE Transactions on Software Engineering, vol. SE-13, no.1, January, 1987.

[MID90] MIDKIFF, S. F. e VAIDYANATHAN, P. *Performance evalutation of communication protocols for distributed processing. Computer Communications*, 13:(5) junho 1990.

[PIL97] PILLA, M. L.; BARRETO, M. E.; SANTOS, R. R.; CAVALHEIRO, G. G. H.; NAVAUX, P. O. A. *Mecanismo de Tolerância a Falhas para a Linguagem de Programação DPC++*. IX SBAC-PAD. Campos do Jordão. SP:SBC, 1997.

[PLA95] PLANK, James S. et alli. *Libckpt: Transparent Checkpointing under Unix*. USENIX Winter 95 Technical Conference. 1995.

[SAN93] SANTOS, R. R.; CAVALHEIRO, G. G. H.; NAVAUX, P. O. A. *Mecanismo de Transporte para Comunicação entre Objetos Distribuídos*. Simpósio Nacional de Redes de Computadores e suas Aplicações. Porto Alegre. SUCESU-RS: 1993.

[SIN94] SINGHAL, Mukesh; SHIVARATRI, Niranjan G. *Advanced Concepts in Operating Systems*. McGraw-Hill. 1994.

[YAU92] YAU, S. S., JIA, X., BAE, D. H.. *Software Design Methods for Distributed Computing Systems*. Computer Communications, 15(4):213-224, May, 1992.

[ZOM96] ZOMAYA, Albert Y.H. *Parallel and Distributed Computing Handbook*. McGraw Hill. 1996.

Reintegração de Servidores em um Sistema de Replicação de Arquivos

Marcia Pasin (pasin@inf.ufrgs.br)
Taisy Silva Weber (taisy@inf.ufrgs.br)

Curso de Pós-Graduação em Ciência da Computação
Universidade Federal do Rio Grande do Sul
Caixa Postal 15064 CEP 91501-970

Resumo

Sistemas distribuídos representam uma plataforma ideal para implementação de sistemas computacionais com alta confiabilidade e disponibilidade devido a redundância fornecida por um grande número de estações interligadas. Falhas em uma estação servidora podem ser contornadas pela reconfiguração do sistema. Entretanto, falhas em seqüência que afetem múltiplas estações comprometem não apenas o desempenho do sistema, mas também a continuidade do serviço e sua confiabilidade. Servidores falhos, que tenham sido isolados do sistema, devem ser reintegrados tão logo quanto possível. Este artigo trata de sistemas de arquivos replicados e da reintegração de servidores nestes sistemas. É assumido um ambiente distribuído que garante alta confiabilidade em aplicações convencionais através da técnica de replicação de arquivos.

Abstract

Distributed systems are an ideal platform to develop high reliable computer applications due to the redundancy supplied by a great number of interconnected workstations. Failed stations can be masked reconfiguring the system. However, sequential faults, that affect multiple stations, not just decrease the performance of the system, but also affect the continuity of the service and its reliability. Thus, failed stations working as servers, that have been isolated from the system, should be reintegrated as soon as possible. This work is about replicated file systems and reintegration of failed servers in this systems. It is assumed a distributed environment that guarantees high reliability in conventional applications through replication of files.

1. Introdução

A replicação de arquivos é uma técnica de tolerância a falhas utilizada para aumentar a disponibilidade e a confiabilidade em sistemas distribuídos. Esta técnica dissemina *cópias de arquivos* entre várias estações servidoras em um sistema. Assim, se uma cópia é perdida acidentalmente, há outra para substituí-la.

Uma abordagem de distribuição destas cópias é a *cópia primária* [BUD 93] Nesta abordagem centralizada, um dos servidores é responsável pela coordenação dos demais servidores. O coordenador contém a cópia primária (e é chamado *servidor primário*) e os demais são *backups* (ou *servidores secundários*). Cada cliente sabe qual servidor é o primário e estabelece comunicação somente com este servidor. Os servidores secundários são apenas repositórios de dados.

Na escrita, o cliente envia o arquivo ao servidor primário. O servidor primário atualiza o arquivo em seu sistema de arquivos e envia uma cópia do arquivo para cada um dos servidores secundários por *difusão de escritas*. Para realizar a leitura de um arquivo, um cliente faz uma

requisição ao servidor primário. O servidor primário realiza a leitura e retorna a informação para o cliente.

Outra abordagem de distribuição é as *réplicas* ou *cópias ativas* [SCH 90]. Esse método submete todas as réplicas às mesmas regras. O controle da replicação não é centralizado como no método de cópia primária. No procedimento de escrita, a invocação do cliente é recebida por todas as réplicas. Cada réplica processa a alteração e retorna a resposta ao cliente. O cliente espera até receber a primeira resposta ou a maioria de respostas idênticas. Na leitura, o cliente faz a invocação e, novamente, espera até que receba a resposta.

O método das réplicas ativas requer que as cópias livres de falhas recebam as invocações dos clientes na mesma ordem. Isso pode ser resolvido através de uma primitiva de comunicação que satisfaça as propriedades de ordenação e de atomicidade.

2. Modelo do Sistema Considerado

Neste artigo considera-se um sistema replicado (por cópia primária ou réplicas ativas) composto por uma rede de estações com poucos servidores e muitos clientes. Os servidores que contém réplicas de um mesmo arquivo formam um *grupo de replicação*. Um *grupo de replicação* fornece os serviços básicos de leitura e escrita para os clientes. Quando um servidor do grupo de replicação falha, os servidores operacionais precisam garantir que a informação do sistema não foi comprometida (isto é, os arquivos indisponíveis possuem cópias no sistema para substituí-los) e continuar o serviço para os clientes. Após a detecção da falha, o servidor deve ser reparado. Então, é necessário reintegrar o servidor à rede.

O sistema considerado tolera apenas um ponto de falha de *crash* por vez em servidor, isto é, apenas um dos *n* servidores pode falhar em dado instante. Quando uma falha é detectada, o servidor deve ser confinado, reparado e reintegrado ao grupo para não comprometer a capacidade de tolerância a falhas do sistema. Apenas falhas em servidores serão consideradas, pois falhas em servidor comprometem todas as estações as quais este servidor presta serviço. Falhas em cliente comprometem apenas uma estação (o próprio cliente).

3. Fases de Operação do Sistema Distribuído Replicado

Pode-se distinguir duas fases na operação de um sistema distribuído com vários servidores: *operação plena* e *operação degradada*; além de duas operações básicas que podem ser realizadas com servidores sob este sistema durante sua vida útil: *reintegração de servidor* e *confinamento de servidor*.

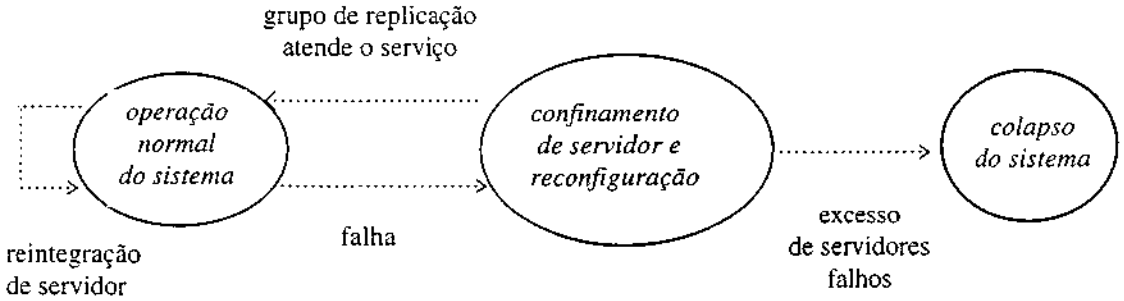


Figura 1 - Fases de operação do sistema distribuído

Durante a *fase normal* (plena ou degradada) (fig. 1), quando uma falha é detectada em um servidor, a estação deve ser *confinada*, tornando seu serviço inacessível. O sistema perde um

servidor e precisa de mecanismos para restaurar o serviço, *reconfigurando* o grupo de replicação. O sistema pode requisitar a substituição ou reparo do servidor perdido.

Depois do *confinamento* do servidor falho e da reconfiguração dos servidores, o sistema retorna a *fase normal* (degradada). Porém, a capacidade de tolerar falhas é reduzida. Um sistema robusto deve suportar determinado número de falhas subsequentes. Esta capacidade está associada ao número de cópias de arquivos disponíveis em cada grupo de replicação.

Para aumentar ou manter a capacidade de um sistema tolerar falhas, o sistema pode sofrer a *reintegração de servidor*. A reintegração começa quando um servidor é integrado fisicamente à rede. Envolve a *atualização do sistema de arquivos do servidor* e a *integração deste com o grupo de replicação*. Para realizar a atualização de suas cópias, o servidor troca mensagens com o grupo de replicação para obter a versão mais recente dos arquivos. Quando o protocolo de atualização termina, o grupo é notificado que o servidor está atualizado e que deverá voltar a participar ativamente das operações do sistema.

4. Reintegração de Servidores

A reintegração de servidores não é assunto facilmente encontrado na literatura. Na maioria das vezes trata-se de um procedimento manual, mas é necessária para prolongar a vida útil do sistema: (a) mantém a atividade plena do sistema, corrigindo uma eventual degradação de desempenho gerada por falha; (b) mantém o número de servidores da configuração original do sistema e (c) aumenta a confiabilidade e disponibilidade de informação, quando um novo servidor é adicionado.

A reintegração de servidor começa quando o servidor é ligado e difunde uma requisição de reintegração para o grupo de replicação ao qual quer se conectar. O grupo responde à requisição e começa o *protocolo de atualização do sistema de arquivo do servidor* (fig. 2).

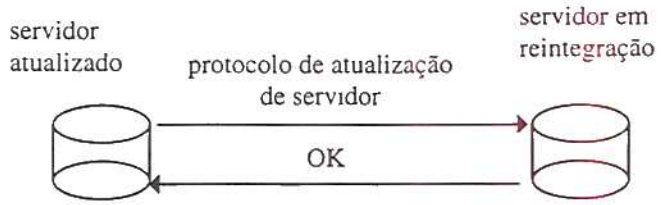


Figura 2 - Protocolo de atualização de servidor

O servidor em reintegração precisa coletar informação do grupo de replicação ao qual quer se integrar para atualizar o seu sistema de arquivos. Quando a atualização terminar, o grupo de replicação deve ser notificado em um procedimento final. São considerados três diferentes protocolos de *atualização de servidor* [LEB 96]: *transferência de volume*[1], *cópia de arquivos* e *retenção de logs*.

A *transferência de volume* é o protocolo mais simples. A partir de dois servidores que decidam que um volume deve ser recuperado, simplesmente o volume atual é copiado completamente para o servidor desatualizado. Um algoritmo de cópia recursiva percorre a árvore de diretórios de um servidor atualizado e transfere todos os arquivos para o servidor desatualizado. Este protocolo é indicado quando um novo servidor é inserido na rede e precisa atualizar todo o seu sistema de arquivos.

[1] conjunto de diretórios de arquivos de um servidor

A *cópia de arquivos* é um protocolo mais restrito que o anterior. Apenas são atualizados os arquivos alterados quando o servidor esteve ausente. Este protocolo é indicado para servidores que ficaram temporariamente desligados. A atualização pode ser realizada com a implementação de um algoritmo recursivo semelhante à transferência de volume, mas que utilize algum critério de comparação, ao invés de realizar a cópia incondicional. O critério da comparação pode ser números de versão associados às cópias dos arquivos. O número de versão inicialmente recebe o valor zero (criação do arquivo ou diretório) e é incrementado após cada operação de escrita realizada.

A *retenção de logs* aloca um espaço no disco de um servidor para servir como *cache* de operações para recuperar servidores falhos. Sempre que uma operação solicitada por um cliente não for transmitida para um servidor em decorrência de falha, será anotada na *cache*. Quando a falha no servidor for corrigida, o servidor com o *log* passa a retransmitir ao servidor que sofreu a falha todas as operações perdidas, tornando-o atualizado. Devido a necessidade de manutenção do *log*, este protocolo deverá ser utilizado para tratar falhas de curta duração.

5. Tornando o NFS Mais Confiável

O NFS (*Network File System*) [SAN 85] é o sistema de arquivos cliente-servidor mais conhecido desde sua implementação na década de 80. A versão atual do NFS comporta muitas necessidades que foram surgindo através de sua ampla utilização, mas ainda não é um sistema completo quando enfocamos alta disponibilidade do serviço. Alta disponibilidade é um determinante crítico, principalmente, para sistemas financeiros.

O protocolo de reintegração de servidor, bem como os métodos de replicação por *software* e *hardware*, podem ser aplicados ao NFS convencional para obter confiabilidade e disponibilidade desejadas. Um exemplo desta tecnologia é o RNFS (*Reliable Network File System*) [LEB 96, LEB 98] que está em desenvolvimento. No RNFS, o NFS convencional foi estendido para suportar replicação de arquivos por *cópia primária* sem utilizar *hardware* especial. O projeto do RNFS prevê a reintegração de servidores ao grupo de replicação. Um protótipo para a reintegração de servidores RNFS está sendo implementado.

6. Prototipação da Reintegração de Servidores

Um protótipo para a atualização de servidores durante a reintegração foi implementado para o RNFS. O protótipo usa o *rpcgen*, que permite a construção de aplicação distribuída com RPC (*Remote Procedure Call*). O cliente da aplicação (servidor secundário que está sendo reintegrado) faz as requisições para o servidor da aplicação (servidor primário). O RNFS assume que o servidor primário conterá sempre uma cópia atualizada de todos arquivos replicados que gerenciar [LEB 96]. Se o primário falhar, um novo primário já foi escolhido dentro do grupo de replicação antes do início da reintegração. Assim, o protótipo considera o servidor primário corrente como fonte de dados incondicional para qualquer protocolo de reintegração.

A parte pronta do protótipo implementa a atualização por *transferência de volume* e por *cópia de arquivos*. Na implementação, o sistema de arquivos de um servidor qualquer é atualizado a partir da informação do servidor primário. A atualização por transferência de volume foi implementada usando as RPCs disponíveis no NFS convencional.

Para implementar a atualização por cópia de arquivos foi necessária a inclusão do número de versão associado a cada arquivo do sistema. Antes de realizar a atualização do arquivo no servidor, a versão do arquivo no primário é consultada. Se a versão do primário para o arquivo

é maior que a versão do servidor que está sendo reintegrado, o arquivo é atualizado. Toda a comunicação adicional entre os servidores envolvidos na atualização é realizada por RPC.

7. Conclusões

Sistemas [BHI 91, LIS 91, LEB 98] foram propostos para tornar o NFS mais confiável. Esses sistemas utilizam replicação de *hardware* [BHI 91] ou *software* [LIS 91, LEB 98] e possuem preocupação especial em prover um ambiente altamente confiável e disponível com detecção automática de falhas, sem alterar a visão convencional do NFS para o usuário. Detectar a falha e confinar o servidor não é suficiente para manter a confiabilidade e disponibilidade esperadas para um sistema. O ideal é que estes sistemas permitam a reintegração automática de servidor.

Um protótipo com os protocolos de atualização foi implementado para realizar a reintegração automática de servidor em um sistema replicado baseado no NFS. Os protocolos usam as RPCs do NFS convencional, além de uma RPC que retorna o número de versão do arquivo para o protocolo de atualização por cópia de arquivos. Para eliminar a necessidade de usar essa RPC adicional, e controlar a desvantagem de tornar o protótipo incompatível com o NFS, uma solução usando a informação que indica a última atualização do arquivo pode ser tentada, se o sistema suportar um algoritmo de sincronização de *clocks*.

Os resultados obtidos pelo protótipo mostraram que, para o mesmo sistema de arquivos, a atualização por cópia de arquivos é mais eficiente que a transferência de volume, exceto se o servidor precisa atualizar todos ou a maioria dos arquivos em seu sistema. Neste caso, o tempo computado para a cópia de arquivos engloba a transferência do volume completo e todas as comparações necessárias para verificar que todo o volume está desatualizado.

Bibliografia

[BHI 91] BHIDE, A., ELNOZAHY, E. N., MORGAN, S. P., A highly available network file server. In: USENIX, 1991. **Proceedings...** [S.l.: s.n.], 1991. p.199-205.

[BUD 93] BUDHIRAJA, N., MARZULLO, K., SCHNEIDER, F., B., TOUEG, S. The primary-backup approach. In: MULLENDER, Sape (Ed.). **Distributed Systems**. 2 ed. New York: ACM Press, 1993. p.199-216.

[LEB 96] LEBOUTE, Mario M. **RNFS - Um sistema de arquivos distribuídos tolerante a falhas para o UNIX**. Porto Alegre: CPGCC da UFRGS, 1996. 85p.

[LEB 98] LEBOUTE, M., WEBER, Taisy S., A reliable distributed file system for UNIX based on NFS. In: IFIP INTERNATIONAL WORKSHOP ON DEPENDABLE COMPUTING AND ITS APPLICATIONS, 1998. **Proceedings...** Johannesburg, 1998. p.158-168.

[LIS 91] LISTOV, Barbara *et al.* Replication in the Harp File System. **Operating System Review**, New York, 1991. v.25, n.5. p.26-238.

[SAN 85] SANDBERG, R., GOLDBERG, D., KLEIMAN, S., WALSH, D., LYON, B. Design and implementation of the Sun Network File System. In: THE SUMMER USENIX CONFERENCE, 1985. **Proceedings...** [S.l.: s.n.], 1985. p.119-130.

[SCH 90] SCHNEIDER, F. B. Implementing Fault-Tolerant Services Using the State Machine Approach: A Tutorial **ACM Computing Surveys,** New York, v.22, n.4, December 1990.

Gerenciamento Confiável de Energia em Redes de Computadores

Mário Magalhães Leboute **Tórgan Flores Siqueira**
Ingrid Jansch-Pôrto

{leboute, torgan, ingrid}@inf.ufrgs.br

Universidade Federal do Rio Grande do Sul – Instituto de Informática

Resumo

O gerenciamento de energia é uma questão de importância crescente em sistemas informatizados, devido ao fato de que as perturbações no fornecimento de energia constituem-se na causa principal de falhas temporárias nos sistemas de informática. Além disso, o consumo total de energia nas redes de computadores é cada vez mais significativo, o que sugere investimentos maiores na otimização de seu uso. Este artigo descreve o trabalho atual no grupo do projeto UFRGS-CP Eletrônica, voltados ao desenvolvimento de soluções distribuídas para o gerenciamento de energia em redes de computadores, integrando os elementos consumidores e fornecedores de energia na rede, e suas aplicações e usuários.

Abstract

Power managing is of growing importance on computer networks as power disturbances are the main reason of temporary faults in unprotected computer systems; the large diffusion of these systems has also contributed for this concern. This paper describes the activities that are in progress on the UFRGS - CP Eletrônica S.A. research project, which aims at the developing of high-integrated power handling solutions for computer networks. Our main goal is to obtain a better power managing by integrating all network elements, power supply and power consumption devices, as well as users and applications inside a globally coordinated model.

1. Gerenciamento de energia e tolerância a falhas

A preocupação com o fornecimento energia é uma questão de importância crescente em redes de computadores. Isto se deve, em primeiro lugar, ao fato de que interrupções no fornecimento de energia são a causa principal de paradas temporárias nestes sistemas. Além da freqüência com que ocorrem, os efeitos da falta de energia podem não se limitar ao período em que esta falta ocorre; a extensão dos reflexos registra-se através do tempo necessário para uma nova partida do sistema, além do risco potencial de perdas de dados e da geração de estados inconsistentes em sistemas de arquivos e bancos de dados.

Uma segunda razão para o aumento da importância do gerenciamento da energia é a busca de economia. A informatização das organizações continua crescendo de modo exponencial, e em diversas organizações o consumo de energia no sistema de informática já supera várias outras fontes importantes de consumo. Desta forma, uma melhora na otimização do uso da energia nas redes pode significar uma economia total significativa, além de contribuir para uma redução no risco de falhas, ao reduzir a pressão sobre o fornecimento.

Em função destes fatos, existe demanda crescente por soluções de gerenciamento de energia para redes de computadores que sejam mais completas do que as atualmente

disponíveis, objetivando tanto o aumento da confiabilidade no fornecimento, como a otimização e racionalização no uso da energia. Os trabalhos no grupo de pesquisa do projeto UFRGS – CP Eletrônica, dirigem-se ao desenvolvimento de soluções para o gerenciamento distribuído do maior número de aspectos possíveis do contexto energético de uma rede local, incluindo todos seus componentes, sejam eles fornecedores de energia (*No-Breaks*) como consumidores (computadores e periféricos).

2. Recursos atuais para gerenciamento de energia

As técnicas atualmente disponíveis para o gerenciamento de energia em sistemas de informática caracterizam-se sobretudo por seu baixo grau de integração. Os fabricantes da linha PC oferecem desde 1991 algumas soluções para a otimização local do uso de energia nestas máquinas, sobretudo em equipamentos portáteis. Os fabricantes de *No-Breaks* eventualmente provém soluções não padronizadas para o gerenciamento local ou remoto das unidades. Nos equipamentos UNIX, o gerenciamento de energia é freqüentemente inexistente.

A família PC apresenta o maior número de recursos disponíveis para gerenciamento de energia. Os primeiros padrões surgiram na década de 1980, destinados a reduzir o consumo em equipamentos que não estavam sendo usados. Com a popularização dos equipamentos portáteis (*"notebooks"*) o gerenciamento de energia ganhou importância, devido à necessidade de otimizar a utilização das baterias. Em 1993 foi formalizada a definição APM 1.0 (*"Advanced Power Managing"*) [INT93], visando normalizar soluções proprietárias que começavam a ser oferecidas pela indústria. Posteriormente, várias características do padrão APM foram transferidas para equipamentos *desktop*, resultando no contexto atual do gerenciamento de energia na família PC. Na prática, devido em parte à introdução tardia do padrão APM, o gerenciamento de energia tornou-se um recurso pouco padronizado. Além disso, são freqüentes as falhas de implementação e implementações parciais do modelo, implicando em que o padrão APM não pode ser considerado de fato como uma norma.

Uma terceira geração para o gerenciamento de energia em PCs está sendo proposta pelo consórcio formado por Intel, Microsoft e Toshiba: a especificação **ACPI 1.0** – *"Advanced Configuration and Power Interface Specification"* [INT96]. A especificação ACPI visa, além de padronizar, estender as capacidades de gerenciamento de energia, e também distribuir os equipamentos em **classes padronizadas** de gerenciamento. As especificações de software do ACPI estão sendo implementadas dentro do Windows98, e fazem parte do projeto "OnNow" do referido consórcio.

Todos os padrões para gerenciamento de energia, incluindo o ACPI 1.0 caracterizam-se por serem soluções locais, sem previsões para uma integração global do gerenciamento de energia.

As deficiências principais nos recursos atualmente disponíveis para gerenciamento de energia podem ser listadas da seguinte forma:

- O gerenciamento é sempre localizado: elementos individuais da rede não levam em consideração o estado de outros elementos na seleção de seus estados de energização.
- Não é possível reunir informação remotamente sobre o estado de energização da maioria dos componentes da rede.
- Não existe suporte para a realização de operações cooperativas e automaticamente coordenadas dentro da rede para mascarar ou reduzir o impacto de falhas no fornecimento de energia.

- Os padrões são incompletos e restritos: não existe a possibilidade de interoperação no gerenciamento de energia entre equipamentos de famílias diferentes.

3. Modelo distribuído para gerenciamento de energia

A busca de melhorias no gerenciamento da energia em redes deve partir do conceito de que a energização é, como vários outros, um recurso compartilhado dentro da rede. A partir deste princípio, todos os componentes da rede devem cooperar para a otimização do uso da energia; para este fim, devem fornecer informações sobre seu próprio uso e eventualmente aceitar comandos remotos sobre energização. Assim, uma camada adicional de controle distribuído de energia pode ser adicionada sobre as camadas locais de controle em cada dispositivo. Esta camada pode implementar uma interface para controle distribuído e mascarar diferenças de padrões nos diversos equipamentos. Este modelo distribuído de controle de energização é o centro da proposta do grupo de pesquisa, e é denominado **NPM** – (*"Network Power Managing"*).

A característica básica do modelo NPM está no princípio da disponibilização de informações e na aceitação de comandos validados sobre energização. Além disto, os recursos de NPM sobrepõe-se, mas não substituem, aos recursos locais de controle de energia. Assim, na ausência de comandos NPM, cada dispositivo ainda pode contar com seu controle local, resultando em aumento da confiabilidade de controle.

Os objetivos que se pretende alcançar com o modelo são:

- Obter a integração das ações de gerenciamento de energia em todos os equipamentos dentro de um contexto global coerente.
- Permitir aos administradores de rede operações centralizadas sobre energia: com a disponibilidade de uma camada de controle integrando todos os dispositivos, os gerentes de rede poderão tomar ações centralizadas do tipo "desligar todas as estações na sala 1", ou "desligar os servidores após as 20:00 horas".
- Permitir o controle da energia por agentes autônomos ou programáveis: através da camada distribuída de controle, parte do gerenciamento de energia pode ser colocado ao encargo de agentes de software programáveis ou "inteligentes", embutidos em aplicações de gerenciamento de redes. Estes agentes podem realizar tarefas rotineiras, ou suprir a ausência do administrador durante uma situação imprevista como, por exemplo, tomar providências relacionadas a uma falta de energia.
- Permitir a supervisão e armazenamento de registros de atividade nos recursos de energia: informações em graus variáveis de detalhe sobre a energização podem ser levantadas, com a observação de tendências e detecção de desperdícios.
- Obter um grau maior de economia, relacionando a energização de alguns equipamentos com o uso global da rede. Por exemplo, servidores e roteadores podem ser colocados em um estado de economia de energia quando um agente centralizado perceber que todas as estações conectadas a eles estão inativas.

3.1. Características do NPM

A estrutura do modelo NPM pode ser vista na figura 1. A característica principal do modelo NPM é a criação de uma camada de software "conector NPM" para acesso remoto aos recurso de gerenciamento de energia disponíveis em cada equipamento

3.2. Protocolos utilizado

Atualmente, são utilizados dois protocolos: o NPM-RPC e o SNMP [RFC1157]. As razões para a existência de dois protocolos são conveniências de desenvolvimento: o NPM-RPC é um protocolo proprietário, baseado em DCE-RPC [DAV95], e destina-se à prototipação rápida de soluções utilizando *"stub compilers"* (no caso MIDL 2.0). Estes compiladores geram código C/C++ a partir de definições escritas em linguagem IDL [RIC89], e permitem a fácil geração de aplicações em rede em modelo cliente-servidor. As aplicações geradas a partir deste modelo são também muito eficientes em sua camada de transporte, e permitem o uso de modalidades limitadas de *broadcasting*. O inconveniente deste protocolo é sua natureza proprietária, que restringe sua utilização apenas a sistemas desenvolvidos dentro do projeto. O protocolo NPM-RPC tem sido usado com propósitos de prototipação, e para a geração de soluções intermediárias requeridas pelo parceiro industrial.

O protocolo SNMP, por outro lado, é um padrão para gerenciamento de redes, e foi escolhido pelo projeto para aplicações finais, devido à sua completa interoperabilidade entre diversos fabricantes, e à sua extensibilidade. Os inconvenientes do SNMP estão na implementação mais complexa, e na necessidade do registro de espaços de variáveis em entidades internacionais de normalização.

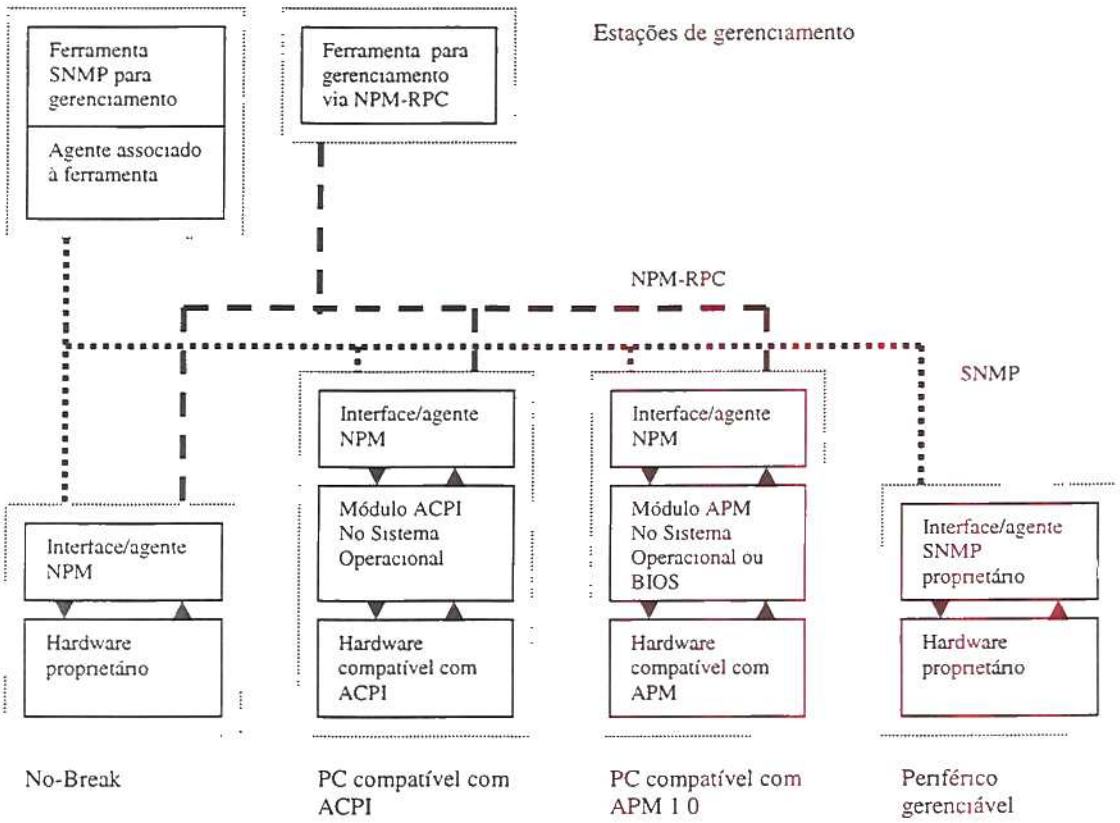


Figura 1: Modelo de gerenciamento NPM

3.3. Bases de informação para gerenciamento de energia

A função dos dois protocolos listados é implementar o transporte de **variáveis de estado**, que refletem ou alteram condições de energização nos dispositivos. A listagem do conjunto das variáveis de estado em um dispositivo define o conjunto de aspectos

gerenciáveis neste dispositivo. Em SNMP, os conjuntos de variáveis de estados são denominadas MIBs (*"Management Information Bases"*).

O padrão NPM define o fluxo de informações de controle para gerenciamento de energia a partir de quatro MIBs padronizadas. Uma delas é a MIB padronizada "UpsMib" [RFC1628], e as outras três estão sendo desenvolvidas dentro do projeto.

Na definição de cada uma das MIBs está sendo utilizada a linguagem padrão ASN.1 [X.680], obrigatória para MIBs SNMP. No caso do uso de NPM-RPC, as especificações tem de ser portadas para IDL.

As MIBs definidas e os tipos de equipamentos a que se destinam estão na tabela 1. Diferenças de capacidade de gerenciamento entre dispositivos de um mesmo grupo são tratadas como variações da disponibilidade de variáveis individuais em cada MIB (algumas variáveis são classificadas como opcionais).

Tipo de dispositivo	MIB
No-Breaks localizados	UpsMib (RFC1628)
Clusters de No-Breaks	UpsCluster1.Mib
PCs compatíveis com padrão APM 1.1 ou superior	RemoteApm1.Mib
PCs e Workstations compatíveis com ACPI	RemoteACPI.Mib
Dispositivos periféricos com MIBS proprietárias	MIBS proprietárias

Tabela 1: Tipos de dispositivos e MIBs associadas

3.4. Controle centralizado

A tomada de ações administrativas no modelo NPM é deixada ao encargo de aplicações genéricas de administração de redes por SNMP. Estas aplicações são altamente configuráveis, e capazes de algum grau de *"inteligência"* como a detecção de tendências e a avaliação de variáveis associadas. A maioria das aplicações de gerenciamento no mercado provê também um *framework* para o desenvolvimento de esquemas de ações automáticas pelos próprios administradores de redes. Está nos objetivos do projeto o teste futuro com estas aplicações.

4. Estado atual

Foram completadas as atividades de desenvolvimento de um software para gerenciamento distribuído de *No-Breaks* produzidos pelo parceiro industrial, via protocolo NPM-RPC, o qual contém também um conector para SNMP suportando a MIB UpsCluster1, o que permite gerenciar estes equipamentos por meio de SNMP.

Está sendo desenvolvida a MIB RemoteApm1, para interface SNMP com os recursos de APM em máquinas Windows 95 e NT. A implementação de agentes de extensão para estes sistemas operacionais, necessários ao suporte às MIBs, estão concluídas. Os testes com o padrão ACPI dependem da obtenção de sistemas operacionais com suporte a este modelo.

Também está sendo estudado de que maneira aplicações padronizadas de gerenciamento de redes podem ser configuradas para o gerenciamento de energia sobre NPM. Prevê-se iniciar a experimentação com os sistemas "HP OpenView" ou "Sun Net Manager".

Créditos

Mário Leboute deve sua participação neste projeto ao financiamento de bolsa DTI no contexto do programa RHAE/CNPq. Tórgan Siqueira é bolsista de Iniciação Científica do CNPq. O projeto *Sistema de controle confiável: uma aplicação em energia* (coordenado por Ingrid Jansch-Pôrto) é financiado parcialmente pela Fundação de Amparo à Pesquisa do Rio Grande do Sul - FAPERGS.

Referências bibliográficas

[DAV95] David Gunter et alli. *Client/Server Programming With Rpc and Dce*. Education & Training, 1995.

[INT93] Intel e Microsoft. *APM 1.1 – Advanced Power Management Specification*. 1993. Documento eletrônico, disponível em "ftp://ftp.intel com/pub/IAL/software_specs/apmv11.doc".

[INT96] Intel, Microsoft e Toshiba. *Advanced Configuration and Power Interface Specification*. 1996. Documento eletrônico, disponível em "http://www.teleport.com/~acpi/spec.htm".

[MAT95] Mathias Hein e David Griffiths; *Snmp Versions 1 & 2: Simple Network Management Protocol - Theory and Practice*. Van Nostrand Reinhold, September 1995.

[RIC89] Richard Snodgrass. *The Interface Description Language: Definition and Use*. Computer Science Press, January 1989.

[X680] International Telecommunication Union (ITU-T). Abstract Syntax Notation One (ASN.1): Specification of Basic Notation (Recommendation X.680). ITU-T, 1994.

Análise de tolerância a falhas no protocolo SNMP

André Peres Ingrid Jansch-Pôrto
{peres, ingrid}@inf.ufrgs.br
Curso de Pós-Graduação em Ciência da Computação
Instituto de Informática - UFRGS
Caixa Postal 15064 - CEP 91501-970
Porto Alegre - RS - Brasil

Resumo

Este artigo apresenta as principais conclusões da análise do protocolo de comunicação SNMP - *Simple Network Management Protocol*, em suas características de funcionamento frente à tolerância a falhas e ao seu uso em sistemas de tempo real.

Sistemas de computação tempo real caracterizam-se por apresentar limites de tempo determinados para a realização das tarefas, sendo estes limites impostos pelo ambiente a ser controlado. É importante ressaltar que neste tipo de sistema, em geral a função de controle se revela como um aspecto crítico.

O SNMP é um protocolo de gerência de redes que se caracteriza por possuir nodos gerentes, responsáveis por coletar e analisar informações dos elementos que formam a rede, e nodos agentes, caracterizados como um conjunto de variáveis que representam o estado atual no nodo. Quando o gerente requisita informações aos nodos, realiza sua monitoração; ao alterar estes valores, exerce controle sobre o nodo.

A possibilidade de utilização do protocolo SNMP em aplicações de tempo real com características de tolerância a falhas depende de parâmetros funcionais do mesmo. Este trabalho analisa as características necessárias para a utilização do SNMP neste tipo de sistema a partir de informações teóricas.

Abstract

In this paper, we present the main results of the analysis of the communication protocol SNMP - Simple Network Management Protocol, considering its fault tolerance and real-time systems operational characteristics.

Real-time computational systems have pre-defined deadlines for tasks accomplishment, being the limits imposed by the environment to be controlled. It is important to stand out that, in this kind of system, in general, control appears as a critical element.

The SNMP protocol is a network management protocol which has management nodes, responsible for collecting and analyzing the information of the network elements; and agent nodes, which are a group of variables that represent the current node state. When the manager requests information to the nodes, these ones are being monitored, when changing these values, the nodes are being controlled.

The use of the SNMP protocol in real-time applications with fault tolerance characteristics depends on factors which are related to the way that this protocol will accomplish its functions. This paper analyzes the necessary characteristics for using SNMP in this kind of system considering theoretical information.

1 Introdução

A gerência de redes é responsável por restabelecer o bom funcionamento de uma rede de computadores em situações problemáticas, através da análise dos

diversos elementos que compõem a rede e da alteração de características destes elementos [ROS96].

Um sistema de gerência de redes deve conter cinco componentes: um ou mais nodos gerenciados, cada um contendo um agente; pelo menos uma estação de gerência de rede, contendo uma ou mais aplicações de gerência; entidades capazes de realizar tanto a tarefa de agente quanto a de gerente; um protocolo de gerência de redes, o qual é utilizado para realizar a troca de informações entre os nodos de gerência e os agentes; e informação de gerência nos agentes.

Um nodo gerenciado é um dispositivo qualquer que possui funções em uma rede, como terminais e computadores de grande porte, passando por impressoras, roteadores, *bridges*, *hubs* ou *modems*.

O principal axioma da gerência de redes, segundo Rose [ROS96], é: "O impacto de adicionar gerência de redes em nodos gerenciados deve ser mínimo, refletindo o mínimo denominador comum". Desta forma, tem-se que as aplicações de gerência de redes devem realizar suas funções nesta grande variedade de equipamentos, interferindo o mínimo possível na operação normal destes dispositivos.

Uma estação de gerência corresponde a um servidor executando o protocolo e aplicações de gerência de redes. Cada nodo gerenciado é visto como um conjunto de variáveis que representam informações referentes ao nodo. Ao disponibilizar estas variáveis à leitura, o nodo permite seu monitoramento e, ao alterar seus valores, o nodo está sendo controlado.

Além das operações de leitura (monitoramento) e escrita (controle), existem as operações: transversal, que permite a uma estação de gerência determinar qual o conjunto de variáveis suportadas pelo nodo; e a de interrupção (*trap*), que informa ao nodo de gerência a ocorrência de alguma exceção na estação gerenciada.

O protocolo de gerência de redes SNMP (*Simple Network Management Protocol*) realiza suas funções entre nodos agentes e gerentes, através de uma base de informações de gerência MIB (*Management Information Base*) composta por um conjunto de variáveis. Estas variáveis ficam disponíveis aos gerentes para consulta ou alteração através da troca de mensagens definidas pelo próprio protocolo.

Para a troca de mensagens, o SNMP utiliza-se do protocolo de transporte não orientado à conexão UDP, o qual é recomendado pelos autores do SNMP como o método mais prático de transporte. Em uma mensagem SNMP, estão contidas as informações referentes ao tipo de mensagem que está sendo enviada (leitura, escrita, transversal ou interrupção), além dos valores referentes à base de informações do agente.

Ao receber uma mensagem de consulta, cabe ao agente fornecer as informações da MIB que atendem à requisição do gerente. Caso a mensagem for de escrita, é realizada uma chamada de procedimento remoto RPC (*Remote Procedure Call*) no agente para efetuar as operações correspondentes. Uma alteração de valor em uma MIB pode ser responsável pela reconfiguração do agente, alterando as características da rede. As mensagens transversais são utilizadas pelo gerente para a obtenção da árvore que compõe a MIB, permitindo recuperar as variáveis que estão disponíveis no agente. Já a mensagem de interrupção ocorre quando é identificado um evento extraordinário no agente. Este agente deve, então, informar ao gerente a ocorrência deste evento. Cabe ao gerente realizar as operações necessárias para tratar uma interrupção.

Durante a comunicação, o protocolo SNMP aguarda a resposta sem interromper suas operações, ou seja, enquanto a resposta a uma requisição não é recebida, o SNMP continua realizando outras tarefas. O SNMP possui um período

máximo de entrega de mensagens, sinalizado pela ocorrência de *time-out*; esta circunstância indica o congestionamento da rede, a queda de um nodo ou a perda de mensagens, possibilitando a identificação de problemas na rede ou nos nodos.

Para realizar a verificação da integridade dos dados que estão sendo trocados entre as entidades SNMP, este protocolo realiza apenas a análise de informações referentes à comunidade. A comunidade é definida como um conjunto de caracteres ASCII que serão utilizados para realizar o relacionamento entre entidades SNMP. Esta análise simples dos dados da comunidade, assim como o tráfego desprotegido das informações representam fraquezas no aspecto de segurança do protocolo, permitindo o ataque de um intruso a informações trocadas entre as entidades.

Com o objetivo principal de solucionar alguns problemas, foi criado o SNMPv2. Esta versão do protocolo possui mecanismos adicionais capazes de contornar os principais problemas relativos à segurança e descrição dos dados, tais como funções de privacidade de dados, autenticação e controle de acesso.

Infelizmente, a segunda versão do protocolo SNMP existe apenas em teoria. Seus criadores encontraram divergências em vários aspectos quanto à forma de implementação dos objetivos (principalmente em relação à segurança) deste protocolo, o que impediu a sua finalização. Em 1995, o grupo de trabalho responsável pelo SNMPv2 foi dividido em subgrupos para estudo de alternativas de modelos de segurança diversos. Em agosto de 1996, foi proposta a resolução destes modelos em um modelo definitivo. Março de 1997 marcou o início dos trabalhos no SNMPv3, retomando as atividade dos grupos que trabalhavam nos modelos do SNMPv2, tendo por principal objetivo resolver os problemas relativos à segurança do protocolo [HAR97].

O presente trabalho objetiva apresentar os principais aspectos da análise do protocolo de gerência de redes SNMP, levando em consideração as características desejáveis para este tipo de protocolo nas áreas de tolerância a falhas e sistemas de tempo real. Os aspectos de segurança mencionados não causam efeito nesta análise, tendo em vista que os sistemas de tempo real considerados como principal alvo de interesse no uso do SNMP têm sido utilizados na automação industrial, onde a rede não possui ligações com o exterior, impossibilitando ataques externos. A versão completa da análise compõe a monografia subscrita pelo autor principal deste trabalho [PER97].

2 Propriedades básicas

Denardin [DEN97] estudou as propriedades necessárias ou características desejáveis para aplicações em tempo real, considerando diversos níveis de implementação na computação (sistemas operacionais, linguagens e protocolos). Foi feita a análise teórica de alguns protocolos, levando em consideração as seguintes características: detecção de erros (corrupção de mensagens, perda de mensagens ou de nodos) entrega de mensagens, flexibilidade, comportamento previsível do protocolo, sincronismo e tolerância a falhas. Este trabalho propõe utilizar as características pré-selecionadas por Denardin para realizar a análise do protocolo SNMP, as quais são brevemente reportadas a seguir com os elementos obtidos naquela referência.

- Detecção de erros

Vários sistemas de tempo real realizam o controle do ambiente (em aplicações críticas, por exemplo) sem intervenção humana. Estes sistemas devem realizar a identificação da ocorrência de erros de maneira ágil, permitindo que o sistema reaja ao erro antes que este seja propagado ao ambiente. Estes protocolos necessitam oferecer suporte adequado para detectar a presença de erros, tais como: a corrupção de

mensagens (mensagens nas quais há perda parcial ou alteração do conteúdo), mensagens perdidas (perda de mensagens previstas no protocolo, ocorridas durante a comunicação); perda do nodos (queda de um ou mais nodos da rede).

- Entrega com tempo determinado

Sistemas de tempo real caracterizam-se pela delimitação do intervalo de tempo para o recebimento de uma informação do ambiente, sua computação e o fornecimento (quando necessário) de respostas ao ambiente. Da mesma forma, os protocolos de comunicação a serem integrados a este tipo de aplicação, devem possuir condições de assegurar intervalos de tempo máximos para realizar a comunicação entre os nodos da rede. Os prazos de entrega de mensagens devem ser conhecidos e limitados.

- Flexibilidade

Esta característica indica se o protocolo é suficientemente flexível para ser utilizado em aplicações diversas ou se foi desenvolvido com limitações a uma arquitetura de rede ou aplicação específica.

O excesso de flexibilidade pode entrar em conflito com a capacidade de detecção de erros em protocolos de comunicação tempo real. A detecção de erros é atingida mais facilmente quando opera sobre um modelo de comportamento rígido, enquanto que esta rigidez traz limitações à flexibilidade.

- Comportamento previsível

A previsibilidade de um sistema de tempo real torna possível a determinação antecipada de seu comportamento. Com previsibilidade permite definir o momento de transmissão de uma mensagem, e o intervalo de tempo necessário à realização deste processo.

Esta característica engloba o comportamento do sistema em relação ao tempo de entrega de mensagens.

- Sincronismo

Esta característica corresponde à análise do suporte oferecido pelo protocolo para sincronização entre os relógios dos nodos. A sincronização é realizada através da comparação entre o instante de tempo em que uma mensagem chega ao nodo destino e o tempo previsto anteriormente para esta chegada.

- Tolerância a falhas

A tolerância a falhas avalia se o nodo é capaz de continuar com suas operações, embora com alguma degradação, após a ocorrência de uma falha. Esta característica é voltada para aspectos adicionais à detecção de erros, tais como a capacidade de realizar atividades para avaliação e confinamento dos danos, recuperação e tratamento dos erros.

As falhas de comunicação: temporais, caracterizadas por atrasos provenientes de uma sobrecarga do sistema; de omissão, que ocorrem devido a erros durante a transmissão; e de particionamento da rede, provenientes de defeitos no meio físico, devem ser consideradas por sistemas de tempo real distribuídos que possuem características de tolerância a falhas.

É importante ressaltar que, assim como comentado com relação ao par flexibilidade - detecção de erros, algumas características apresentadas podem ser conflitantes entre si, ou seja, um protocolo pode apresentar fortemente alguma característica que implique na redução ou inexistência de outra. Um outro conflito possível, por exemplo, é entre a utilização de técnicas que implementem tolerância a falhas e/ou detecção de erro, ocasionando atrasos na entrega de respostas do sistema, prejudicando conseqüentemente a entrega das mensagens nos prazos-limite do sistema.

3 Análise das características do SNMP

Tendo em vista o emprego do protocolo SNMP em operações de gerência em redes de computadores e em sistemas de automação industrial, ele também se torna um alvo de interesse para análise de acordo com a ótica apresentada por Denardin. Esta análise não traz resultados definitivos, mas pode ser considerada como base para a especificação e implementação de mecanismos que venham a corrigir deficiências detectadas pelo procedimento.

3.1 Detecção de erros

A detecção de erros pelo protocolo UDP é feita através da verificação do *checksum* contido em seus pacotes. O controle de mensagens corrompidas é realizado através da garantia de que caso o pacote UDP não for recebido por completo, ou se o cálculo de *checksum* não conferir com o seu conteúdo, a mensagem será descartada pelo protocolo; logo, apenas as mensagens corretas (completas e com cálculo de *checksum* correto) serão consideradas pelo protocolo SNMP. Apesar de identificar a corrupção da informação contida no pacote, nenhuma notificação será gerada ao protocolo SNMP.

As mensagens perdidas, particionamento da rede, assim como a perda de nodos, são identificadas pela aplicação de gerência através da ocorrência de *time-outs*. Caso a resposta a uma requisição não seja entregue no prazo determinado, a aplicação de gerência irá realizar as operações previstas de retransmissão de mensagens ou reconfiguração da rede.

Através da análise das informações recebidas pelos agentes, a aplicação de gerência é capaz de identificar uma série de problemas referentes ao nodo analisado podendo, assim, realizar operações para solucionar ou contornar estes problemas. É através destas ações que o SNMP permite às demais aplicações que utilizam a rede permanecerem em funcionamento.

3.2 Entrega com tempo determinado

O protocolo SNMP aguarda pelas respostas de suas requisições sem interromper suas operações. Este protocolo possui um intervalo de *time-out* definido pela aplicação, que determina o período máximo de tempo de espera pelo transporte de uma mensagem.

Através da utilização deste tempo máximo previsto para a troca de mensagens, é possível à aplicação determinar a ocorrência das situações de comunicação que excedam o prazo determinado, em decorrência de congestionamento da rede, perda de algum nodo ou perda da mensagem.

A determinação do tempo máximo para a troca de mensagens depende da configuração e dos equipamentos utilizados na rede. Com a construção de uma rede de dispositivos e a análise do tempo na troca de mensagens entre eles, é possível construir um sistema previsível que utilize o protocolo SNMP, respeitando os limites de tempo impostos por sistemas de tempo real. Este sistema irá identificar a ocorrência de atrasos na transmissão de mensagens entre os nodos através do *time-out*, possibilitando a identificação de falhas no sistema de tempo real.

3.3 Flexibilidade

O protocolo SNMP tem suas funções objetivamente dirigidas à gerência de redes de computadores, através da consulta e alteração de valores em bases de dados específicas (as MIBs) que se encontram distribuídas pelos nodos da rede. Não existe qualquer restrição explícita quanto à utilização do protocolo SNMP para outras aplicações.

Em relação ao conflito existente entre a detecção de erros e a flexibilidade dos protocolos, temos que a detecção de erros está baseada na forma simples como o

SNMP realiza suas funções, o que permite a previsibilidade de seu comportamento e a identificação de qualquer situação divergente com relação a esta previsão. Já a flexibilidade, é garantida pela maneira como são realizadas as operações de consulta e alteração de valores, possibilitando a utilização do protocolo SNMP em diversos tipos de sistemas.

3.4 Comportamento previsível

Apesar de possuir um meio de transporte não orientado à conexão, o SNMP tem condições de identificar a ocorrência de problemas de perda de mensagens e nodos, através da utilização do tempo máximo de transporte de mensagens. Além disso, a forma simples como este protocolo foi implementado, e a capacidade de receber informações referentes aos equipamentos que formam a rede, permite ao protocolo prever o comportamento destes equipamentos.

O protocolo possui condições de analisar o estado em que se encontram os nodos pertencentes à rede, podendo compor uma visão geral do estado da rede. Isto possibilita a identificação de problemas e a reconfiguração necessária para contorna-los.

3.5 Sincronismo

Apesar de não utilizar sincronismo na sua definição devido à necessidade de acréscimo de tarefas aos nodos agentes para realizar este tipo de controle, o protocolo SNMP apresenta condições para a implementação desta característica.

Para adicionar sincronismo nos nodos que utilizam o protocolo SNMP na gerência de redes de computadores, é necessário o acréscimo de funções de sincronismo aos nodos agentes da rede. Entretanto, o acréscimo de funções nos nodos agentes deve ser controlado para evitar que as funções básicas destes nodos sejam prejudicadas pela gerência.

3.6 Tolerância a falhas

O gerente possui uma visão do estado dos nodos agentes, através da requisição de informações destes nodos. Isto possibilita a identificação de problemas nos dispositivos gerenciados e da ocorrência de congestionamento em determinado ponto da rede, além de outros problemas específicos de configuração dos equipamentos que compõem a rede os quais serão analisados e contornados pela aplicação de gerência. Desta forma, cabe ao SNMP identificar os problemas da rede.

Quando um gerente detecta um nodo com problemas, cabe à aplicação de gerência realizar a reconfiguração da rede de maneira a mascarar ou confinar o erro encontrado. Tem-se então que a aplicação de gerência é a responsável por realizar operações para solucionar os problemas nos sistemas que utilizam SNMP, considerando as características do transporte de mensagens realizadas por este protocolo.

O protocolo utiliza o *time-out* como identificador de falhas temporais e omissão, mas a decisão de retransmissão de mensagens cabe à aplicação.

4. Conclusões

O gerenciamento de redes de computadores é uma forma de acrescentar confiabilidade. A função da aplicação de gerência é a de manter a rede funcionando adequadamente. A aplicação mantém uma visão global da rede, cuidando para que cada componente execute suas funções corretamente.

A gerência então realiza funções de tolerância a falhas para o funcionamento correto da rede, utilizando o conceito de mascaramento de falhas, quando necessário.

Quando não é possível à aplicação realizar suas funções de reconfiguração, é esperado que o usuário receba de maneira imediata a notificação da falha encontrada.

Segundo [BIR96], um protocolo que implemente um canal de comunicação confiável deve garantir, em princípio, que mensagens perdidas serão retransmitidas e que mensagens fora de seqüência serão reordenadas e enviadas na ordem original. O controle de fluxo e mecanismos que inibem o remetente, quando o volume de dados torna-se excessivo, também são comuns em protocolos para transporte confiável. O protocolo UDP não pode ser considerado como um protocolo confiável, devido às suas características de transmissão de mensagens.

A escolha desta forma de transporte pelos autores do SNMP deve-se ao objetivo principal da aplicação de gerência através deste protocolo: realizar operações de análise e alteração da configuração da rede inclusive quando esta se encontra nos piores estados, utilizando o mínimo de recursos e causando o mínimo impacto sobre as funções normais dos nodos.

Desta forma, tem-se que a melhor maneira de se efetuar gerência de redes com confiabilidade, é através do desenvolvimento de uma aplicação com técnicas de tolerância a falhas que utilizem a simplicidade da implementação do modelo do protocolo SNMP de maneira a não sobrecarregar os nodos envolvidos no gerenciamento, e considerando a falta de recursos de confiabilidade existentes no protocolo de comunicação UDP.

A construção de transmissão confiável pode ser alcançada através da utilização das informações de identificação das mensagens (para garantir a ordem), e os controles de perda de mensagens através da verificação dos tempos máximos de resposta. Estas operações devem ser realizadas através de mensagens SNMP, na aplicação de gerência.

O protocolo SNMP apresenta implementação simples, e características de flexibilidade e previsibilidade que tornam sua utilização viável nos mais diversos tipos de sistemas. Sistemas de tempo real podem realizar suas funções utilizando-se das características de troca de mensagens, estrutura de dados e simplicidade de modelo e funcionamento do protocolo SNMP.

5. Referências

[BIR96] Birman, Kenneth P. *Building Secure and Reliable Network.* Applications. Manning Publications Co. Greenwich, CT, 1996.

[DEN97] Denardin, Fernanda Kruel. *Software tolerante a falhas para aplicações tempo real.* Dissertação (mestrado) CPGCC-UFRGS, 1997.

[PER97] Peres, André. *Análise de tolerância a falhas no protocolo SNMP.* Trabalho Individual CPGCC-UFRGS, 1997.

[ROS96] Rose, Marshall T. *The Simple Book - An Introduction to Networking Management.* Revised Second Edition. Englewood Cliffs: Prentice-Hall, Inc. Upper Saddle River, NJ. 1996.

[HAR97] Harrington, David. *The Evolution of Architectural Concepts in the SNMPv3 Working Group.* The Simple Times - The Quarterly Newsletter of SNMP Technology, Comment, and Events. Volume 5, N.1 Dezembro, 1997. Disponível por www em http://www.simple-times.org.

Um Esquema de Injeção de Defeitos Baseado em Operadores de Mutação

Elisa Yumi Nakagawa
José Carlos Maldonado

ICMSC – USP
Av. Dr. Carlos Botelho, 1465
C. Postal· 668, CEP: 13560-970
São Carlos - SP, Brasil
{elisa, jcmaldon}@icmsc.sc.usp.br

Resumo

A Injeção de Defeitos (*Fault Injection*) é uma técnica que vem sendo utilizada para a construção de sistemas que precisam ser altamente confiáveis. Para a sistematização dessa técnica, a utilização de taxonomias ou modelos de defeitos tem um papel fundamental. Neste artigo é proposto um esquema de injeção de defeitos baseado nos operadores de mutação do critério de teste Análise de Mutantes.

Abstract

Fault Injection has been used for high dependable system development. The use of a taxonomy or a fault model is fundamental for making it a systematic activity. In this article, a fault injection scheme based on mutation operators from Mutation Analysis testing criterion is presented.

1. Introdução

Na área de Tolerância a Defeitos (*Fault Tolerance*), a Injeção de Defeitos (*Fault Injection*) é uma técnica emergente que tem sido empregada para a construção de sistemas que precisam ser altamente confiáveis [CLA95]. De um modo geral, essa técnica envolve a injeção controlada de defeitos e a observação do aspecto comportamental do sistema frente à presença dos defeitos.

Para auxiliar na sistematização da atividade de Injeção de Defeitos, tornando-a mais objetiva e eficaz, são requeridos taxonomias ou modelos de defeitos que representem os defeitos a serem injetados. Assim, dentro desse contexto, o estudo e o estabelecimento de taxonomias ou modelos de defeitos tornam-se bastante relevante.

Dentre os trabalhos sobre Injeção de Defeitos, existem estudos que englobam defeitos de software e de hardware. Observa-se que existem poucos trabalhos relacionados à injeção de defeitos de software na literatura [SOM97]. Segundo Martins [MAR93] isso ocorre devido à falta de modelos de defeitos representativos e amplamente aceitos, além da existência de um número limitado de mecanismos visando à tolerância a defeitos de software. Além da falta de modelos de defeitos de software, não existe na literatura um consenso sobre o que seria, exatamente, um modelo de defeitos de software e quais itens — tipo, freqüência, criticalidade dos defeitos, entre outros — deveriam estar presentes no modelo.

Observa-se que existem diversos trabalhos na literatura, como o de Basili/Perricone [BAS84], de Beizer [BEI90], de DeMillo/Mathur [DEM95], entre outros, que visam à classificação dos defeitos encontrados nas diversas fases do ciclo de vida de desenvolvimento do software. Essa diversidade deve-se ao fato da classificação dos defeitos ser uma tarefa difícil, pois os defeitos podem ser classificados considerando-se vários fatores: o comportamento do programador, o efeito causado pelo defeito no programa, entre outros.

Esses estudos enfatizam a relevância do estabelecimento de modelos de defeitos de software concretos e que representem os defeitos "reais" que ocorrem em um sistema de software. O estudo sobre modelos de defeitos de software é ainda um ponto em aberto do ponto de vista da pesquisa científica, diferente dos modelos de defeitos de hardware, sobre os quais pode-se identificar diversos trabalhos.

Atualmente, a injeção de defeitos de software mostra-se relevante por questões de qualidade e produtividade, por isso um dos objetivos deste trabalho é abordar a injeção de defeitos em sistemas de software, mais especificamente no programa fonte do sistema. Assim, na Seção 2 é apresentado o esquema de injeção de defeitos de software e na Seção 3, a ferramenta de injeção de defeitos implementada.

2. Um Esquema de Injeção de Defeitos

Neste trabalho, foi estabelecido um esquema de injeção de defeitos de software baseado na taxonomia de defeitos de DeMillo/Mathur [DEM95] e nos operadores de

mutação [AGR89] do critério de teste denominado Análise de Mutantes [BUD81, DEL93].

Na taxonomia de DeMillo/Mathur é definido um total de quatro classes com características bem definidas baseadas em transformações sintáticas. Na Tabela 1 são apresentadas as classes e as correspondentes porcentagens; por exemplo, 53,26% do total de defeitos no código de um programa são pertencentes à classe Defeito de Entidade Simples Incorreta.

Classe	%
Defeito de Entidade Simples Incorreta (*Simple Incorrect Entity Fault*)	53.26
Defeito de Falta de Entidade (*Missing Entity Fault*)	40.55
Defeito de Entidade Espúria (*Spurious Entity Fault*)	0.69
Defeito de Entidade Mal Empregada (*Misplaced Entity Fault*)	5.50

Tabela 1: Classes da Taxonomia de DeMillo/Mathur

Para a definição do esquema de injeção de defeitos foi realizado o mapeamento da taxonomia de defeitos de DeMillo/Mathur para os operadores de mutação do critério de teste Análise de Mutantes. Esses operadores têm a função de realizar transformações sintáticas no código do programa em avaliação. O mapeamento não é uma tarefa trivial, uma vez que muitos dos operadores possuem comportamento bastante variados dependendo do código no qual está sendo aplicado.

O grupo de pesquisa do ICMSC/USP tem desenvolvido trabalhos relacionados aos critérios de teste de software, incluindo o critério Análise de Mutantes. Utilizando-se da experiência adquirida, os operadores de mutação estão sendo utilizados como mecanismo de geração de defeitos (transformações sintáticas no código) de software, uma vez que retratam os defeitos mais comuns cometidos pelo programador ao longo do processo de desenvolvimento de software [DEM78].

Esses operadores foram definidos por Agrawal e totalizam 71 operadores divididos em quatro classes: Mutação de Comandos (15 operadores), Mutação de Operadores (46 operadores), Mutação de Constantes (3 operadores) e Mutação de Variáveis (7 operadores) [AGR89].

Realizado o mapeamento, para a classe Defeito de Entidade Simples Incorreta foram relacionados 59 operadores, para a classe Defeito de Falta de Entidade relaciona-se dois operadores, para a classe Defeito de Entidade Espúria tem-se nove operadores e para a classe Defeito de Entidade Mal Empregada, um operador.

Como exemplo do mapeamento, considere o operador SSDL (*Statement Deletion*), pertencente à classe Mutação de Comandos, que tem a função de apagar comandos no código do programa, retratando defeitos de falta de comandos. Esse operador foi então relacionado à classe Defeito de Falta de Entidade da taxonomia adotada. Um outro exemplo é o operador ORRN (*Operator Relational Relational Non-assignment*), pertencente à classe Mutação de Operador, que troca um operador relacional por outro relacional, representando defeitos de operadores relacionais empregados incorretamente.

Por realizar esse tipo de mudança no código, esse operador foi classificado como Defeito de Entidade Simples Incorreta.

Durante a atividade de injeção de defeitos, a porcentagem relacionada a cada uma das classes é considerada no cálculo do número de defeitos a ser injetado, ou melhor, ocorre uma seleção dos defeitos gerados pelos operadores de modo que o número de defeitos a serem injetados corresponda à porcentagem de cada classe a que pertencem esses operadores.

3. Ferramenta de Injeção de Defeitos

Dada a crescente complexidade dos sistemas computacionais, a atividade de injeção de defeitos é inviável sem a disponibilidade de uma ferramenta que automatize tal atividade; dessa forma, uma ferramenta de injeção de defeitos, denominada ITool, foi implementada. Tal automatização reduzirá os defeitos conseqüentes da intervenção humana, bem como o tempo e os custos necessários se essa atividade fosse realizada manualmente, quando possível.

Essa ferramenta apresenta uma interface gráfica construída em Tcl/Tk [WEL95]. Na Figura 1 é apresentada a janela principal dessa ferramenta e na Figura 2 a janela que permite ao usuário a ativação/desativação de defeitos no processo de injeção de defeitos.

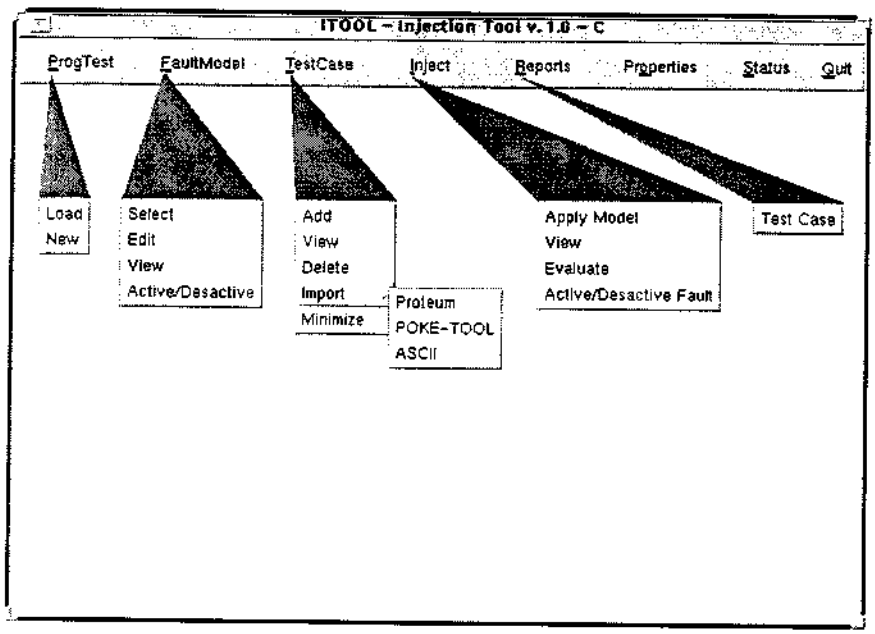


Figura 1: Janela Principal da ITool

A injeção de defeitos gerados pelos operadores de mutação não é uma tarefa trivial, uma vez que ocorrem problemas de priorização de aplicação de defeitos se dois ou mais defeitos relacionam-se a um mesmo ponto de aplicação. Esses e outros problemas são discutidos em mais detalhes em [NAK98].

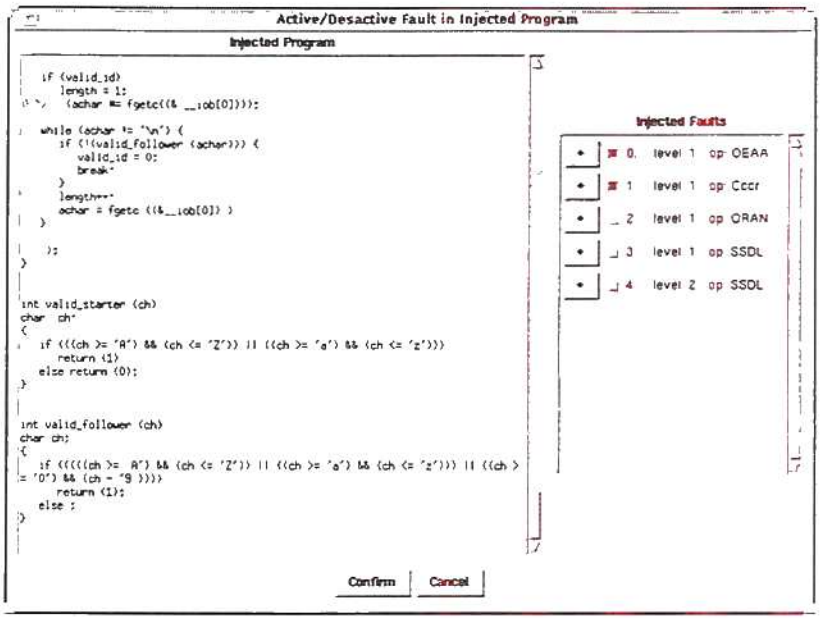


Figura 2: Janela para Ativação/Desativação de Defeitos

De maneira geral, a ferramenta trabalha com sessões de teste (opção *ProgTest* do menu da janela principal apresentada na Figura 1) que são caracterizadas por um conjunto de informações, como por exemplo, o nome do arquivo fonte e do executável do programa sendo avaliado, o conjunto de casos de teste e os resultados alcançados. A ferramenta ainda permite a manipulação de casos de teste (*TestCase*), além da manipulação do modelo de defeitos (*FaultModel*), bem como do programa com defeitos injetados (*Inject*). Permite ainda a impressão de relatórios (*Report*) e acompanhar o andamento da avaliação dos defeitos injetados (*Status*).

Com relação à manipulação do programa com defeito injetados, a ferramenta permite injetar, ativar/desativar defeitos, visualizar o programa com defeitos injetados, além de permitir a edição do programa com defeitos injetados, quando o usuário identifica um defeito "real" no programa.

Atualmente, a ITool está sendo empregada em um estudo de caso utilizando-se o programa Space que trata-se de um sistema real desenvolvido pela ESA (European Space Agency). Aplicando-se o esquema de injeção de defeitos estabelecido com uma taxa de 10 FKLOC (10 defeitos a cada 1000 LOC´s), tem-se a injeção de 37 defeitos divididos em: 20 defeitos da classe Defeito de Entidade Simples Incorreta, 15 da classe Defeito de Falta de Entidade, nenhum defeito da classe Defeito de Entidade Espúria e dois da classe Defeito de Entidade Mal Empregada.

4. Conclusão

As taxonomias ou modelos de defeitos têm um papel fundamental no processo de desenvolvimento de software, pois podem retratar dos defeitos que podem a vir ocorrer durante o processo de desenvolvimento do software, permitindo assim que desenvolvedores concentrem suas atenções em determinadas partes — requisitos,

especificação, código, entre outros — do software.

No tocante à Injeção de Defeitos, os modelos de defeitos podem auxiliar na sistematização dos defeitos a serem injetados. Assim, esse trabalho concentra-se no estabelecimento de um esquema de injeção de defeitos baseado nos operadores de mutação do critério de teste Análise de Mutantes, utilizando-se como base a taxonomia de defeitos definida por DeMillo/Mathur.

Atualmente, a ferramenta ITool está sendo utilizada em um estudo de caso utilizando-se o programa Space para mostrar a factibilidade da utilização de injeção de defeitos de software baseado no esquema de injeção de defeito e no mapeamento sobre os operadores de mutação realizado.

Referências

[AGR89] Agrawal, H.; DeMillo, R.A.; Hathaway, B.; Hsu, W.; Hsu, Wy.; Krauser, E.W.; Martin, R.J.; Mathur, A.P.; Spafford, E.; *Design of Mutant Operators for the C Programming Language*, Technical Report SERC-TR-41-P, Software Enginnering Research Center, Purdue University, março 1989.

[BAS84] Basili, V.R.; Perricone, B.T.; *Software Errors and Complexity: An Empirical Investigation*, Communications of the Acm, V.27, N.1, janeiro 1984.

[BEI90] Beizer, B.; *Software Testing Techniques*, 2ª Edição, Van Nostrand Eeinhold, New York, 1990.

[BUD81] Budd, T.A.; *Mutation Analysis: Ideas, Examples, Problems and Prospects*, Computer Program Testing, North-Holand Publishing Company, 1981.

[CLA95] Clark, J.A.; e Pradham, D.K.; *Fault Injection - A Method for Validating Computer-System Dependability*, IEEE Computer, V.28, N.6, junho 1995, p.47-56.

[DEL93] Delamaro, M.E.; *Proteum - Um Ambiente de Teste Baseado na Análise de Mutantes*, Dissertação de Mestrado, ICMSC/USP, São Carlos, SP, outubro 1993.

[DEM78] DeMillo, R.A.; *Software Testing and Evaluation*, The Benjamim/ Commings Publishing Company, Inc, 1978.

[DEM95] DeMillo, R.A.; Mathur, A.P.; *A Grammar Based Fault Classification Scheme and its Application to the Classification of the Errors of TEX*, novembro 1995 (correspondência pessoal).

[MAR93] Martins, E.; *Validação Experimental da Tolerância a Falhas: A Técnica de Injeção de Falhas*, V Simpósio de Computadores Tolerantes a Falhas, São José dos Campos, SP, outubro 1993.

[NAK98] Nakagawa, E.Y.; Maldonado, J.C.; *Estudo e Priorização da Aplicação dos Operadores de Mutação para Injeção de Defeitos*, Relatório Técnico, ICMSC/USP, São Carlos, SP, março 1998 (em preparação).

[SOM97] Somani, A.K.; Vaidya, N.H.; *Understanding Fault Tolerance and Reliability*, IEEE Computer, V. 30, N. 4, abril, 1997.

[WEL95] Welch, B.B.; *Practical Programming in Tcl and Tk*, Prentice Hall, 1995.

Utilizando metaobjetos para injetar falhas e monitorizar seus efeitos

Amanda Cibele Apolinário Rosa*, Eliane Martins
Instituto de Computação - Universidade Estadual de Campinas (Unicamp)
P.O Box 6176 Campinas - 13083-970 SP Brasil
{amandapo, eliane}@dcc.unicamp.br

Abstract

Software-implemented fault injection is, nowadays, a largely used technique to validate dependability properties of software systems. To inject faults and monitor their effects some form of instrumentation may be introduced into the system under test (target system). This instrumentation causes some level of intrusiveness, i.e., it imposes some interference upon the target system execution. Therefore, a goal of a software instrumentation approach is to be the least intrusive possible. To obtain this quality we propose in this study the use of reflective object-oriented programming. Intrusiveness in the target system is reduced, in that it allows a clear separation between functional and non-functional aspects, the later being related to fault injection and monitoring aspects. This papers describes a reflective test architecture, shows how faults are injected using reflection and presents some results of experiments.

Resumo

Injeção de falhas por software é uma técnica que vem sendo muito utilizada para validar as propriedades de segurança no funcionamento de sistemas de software. Para injetar falhas e monitorizar seus efeitos alguma forma de instrumentação deve ser introduzida na aplicação em teste (aplicação alvo). Essa instrumentação é intrusiva, ou seja, interfere na execução da aplicação alvo. No entanto, um dos objetivos de uma abordagem de instrumentação de software é ser o menos intrusiva possível. Para alcançar este objetivo neste estudo nós propomos o uso da programação orientada a objetos reflexiva. Reflexão reduz a interferência na aplicação alvo porque provê uma separação clara entre os aspectos funcionais e não-funcionais, sendo o último relacionado aos aspectos de injeção de falhas e monitorização. Este artigo descreve uma arquitetura reflexiva de teste, mostra como injetar falhas utilizando reflexão e apresenta alguns resultados de experimentos.

* Trabalho parcialmente financiado pela FAPESP.

1. Introdução

Injeção de falhas por software é uma técnica que vem sendo muito utilizada para validar as propriedades de segurança no funcionamento de sistemas de software e diversas ferramentas para tal propósito têm sido desenvolvidas. Tais ferramentas utilizam abordagens diferentes para realizar a injeção de falhas, algumas dessas abordagens são: inserção de instruções de *trap* (exceções de software) no código fonte alvo, execução da aplicação alvo no modo traço ou no modo privilegiado e utilização de características especiais do hardware.Uma boa apresentação da área é feita em [HTI97]. Apesar do grande número de ferramentas, a injeção de falhas por software tem suas limitações. Uma delas é a perturbação na aplicação em teste: a fim de injetar falhas e monitorizar seus efeitos alguma forma de instrumentação é introduzida no código fonte da aplicação alvo. Essa instrumentação interfere na execução da aplicação e/ou altera a estrutura original do seu código fonte. Em nosso trabalho nós propomos o uso de reflexão computacional para minimizar a perturbação no software da aplicação alvo nos testes de injeção de falhas por software em aplicações orientadas a objetos. Reflexão foi escolhida por ser uma maneira simples de separar a funcionalidade da instrumentação da funcionalidade da aplicação alvo. Reflexão introduz um modelo de arquitetura no qual existe o nível base e o meta-nível. O nível base neste estudo é usado para implementar objetos da aplicação alvo. O meta-nível permite a programadores observar e manipular dados e ações executadas no nível base e neste estudo é utilizado para implementar as características de injeção de falhas e monitorização.

O restante do texto está organizado da seguinte forma: a seção 2 apresenta rapidamente uma visão sobre programação orientada a objetos reflexiva e suas vantagens para injeção de falhas por software, enfocando a linguagem usada neste estudo, OpenC++ 1.2. A seção 3 descreve nossa arquitetura de teste e apresenta como a abordagem metaobjetos é utilizada para injeção falhas e monitorização. A seção 4 apresenta alguns resultados de experimentos realizados. E finalmente a seção 5 conclui este artigo.

2. Reflexão e orientação a objetos

2.1 Conceito e vantagens

Reflexão Computacional, ou apenas reflexão, é a atividade executada por um sistema computacional quando faz computações sobre suas próprias computações [Mae87]. Reflexão é obtida subdividindo-se o sistema em dois níveis de processamento: o *nível base*, que implementa a funcionalidade da aplicação e o *meta-nível*, que observa e manipula a estrutura e/ou comportamento do nível base.

Patti Maes [Mae87] introduziu a abordagem *metaobjeto* para implementar reflexão em sistemas orientados a objetos. Um objeto x pode ser associado a um metaobjeto ^x que representa a meta-informação de x.

Reflexão e orientação a objetos apresentam diversas vantagens para injeção de falhas por software:

1. Redução da perturbação no código da aplicação alvo. Os requisitos de injeção de falhas e monitorização (encapsulados em meta-objetos) ficam separados dos requisitos associados ao propósito da aplicação (encapsulados em objetos).
2. Reutilização de componentes. A separação de domínios permite que os aspectos de injeção de falhas e monitorização sejam desenvolvidas independentes da aplicação, propiciando a reutilização tanto de objetos da aplicação como de metaobjetos.

3. Flexibilidade na injeção de falhas. O uso de metaobjetos possibilita injetar diferentes tipos de falhas em diferentes objetos do nível base, de acordo com suas características.
4. Facilidade de uso. Metaojetos de injeção de falhas e monitorização podem ser facilmente incorporados e retirados da aplicação em teste.

2.2 A linguagem OpenC++ 1.2

OpenC++ 1.2 [Chi93], uma extensão de C++ que dá apoio à reflexão, é a linguagem que vem sendo utilizada neste trabalho. Suas principais características são [Chi93, Lis97]:

1. Objetos no nível base podem ser reflexivos ou não. Objetos reflexivos podem possuir métodos e atributos reflexivos e são associados a metaobjetos, os quais controlam seu comportamento. Um objeto não-reflexivo é um objeto C++ convencional.
2. Cada objeto reflexivo pode ser associado a um único metaobjeto.
3. Um metaobjeto controla a invocação de métodos reflexivos e o acesso a atributos reflexivos do objeto reflexivo do nível base a ele associado.
4. Um metaobjeto é uma instância de uma classe de meta-nível. Classes de meta-nível são associadas a classes do nível-base em tempo de compilação.
5. Classes do meta-nível são derivadas da classe predefinida *MetaObj*, a qual define os métodos para um metaobjeto controlar o objeto reflexivo a ele associado.
6. Um nome_de_categoria pode ser associado a métodos e atributos reflexivos, permitindo que um metaobjeto altere tais métodos e atributos de acordo com a categoria especificada.
7. Um programa em OpenC++1.2 é um programa C++ que contém diretivas para declarar classes, atributos e métodos como reflexivos. Essas diretivas são incorporadas como comentários C++ que começam com *//MOP*.

Essas características ficarão mais claras na próxima seção com os exemplos mostrando seu uso na arquitetura proposta.

3. Utilizando programação orientada a objetos reflexiva para injeção de falhas e monitorização

Nesta seção fazemos uma descrição rápida da nossa ferramenta; apresentamos a arquitetura reflexiva e descrevemos como é feita a injeção de falhas e a monitorização de seus efeitos.

3.1 Arquitetura

A arquitetura para injeção de falhas e monitorização é apresentada na figura 1. Seus principais componentes são: uma *aplicação alvo* com os mecanismos de tolerância a falhas a serem validados, uma *biblioteca de meta-nível*, um *controlador* e um *módulo de interface com o usuário*.

A *aplicação alvo* é uma aplicação tolerante a falhas orientada a objetos que deve ser submetida a uma fase de instrumentação. Nessa fase são introduzidas as diretivas de reflexão para indicar quais objetos e seus respectivos métodos e atributos serão reflexivos, ou seja, serão injetados e/ou monitorados.

A *biblioteca de meta-nível* deve ser anexada ao sistema alvo em tempo de compilação e define metaobjetos denominados *escalonadores*; um escalonador é composto por um *injetor* e *sensor*[1]. Metaobjetos *escalonadores* capturam a mensagem (chamada de método ou acesso a

[1] O escalonador é então um objeto composto, criado devido a uma limitação da linguagem OpenC++ 1.2: cada objeto pode ser associado a um único metaobjeto. A separação das funções de injeção e monitorização é necessária para fins de modularidade e reusabilidade, justificando assim a necessidade de um objeto composto.

atributo) enviada a um objeto reflexivo, verificam a categoria (monitorização e/ou injeção) associada a mensagem, decidem que tipo de ação deve ser realizada, injeção de falhas, coleta de dados ou computação normal do objeto do nível base e podem delegar operações a um *injetor* ou *sensor*. Um *injetor* corrompe valores do objeto base. Cada *sensor* coleta dados sobre a ativação dos mecanismos de tolerância a falhas e sobre o valor de argumentos e atributos de objetos da aplicação definidos para ser monitorizados, além disso coleta dados sobre o processo de injeção de falhas.

O *controlador* coordena os experimentos: inicia o processo da aplicação, controla (via *escalonador*) *injetores* e *sensores* e armazena os dados coletados. O controlador é composto por dois objetos: um *gerente de injeção* e um *monitor*. O *gerente de injeção* lê a falha ser injetada do arquivo *falhas* e a transfere ao *escalonador* que por sua vez a transfere ao *injetor*. O *monitor* recebe os dados enviados pelos *sensores* e os armazena no arquivo de *histórico* do experimento.

A *interface do usuário* auxilia os testadores a observar e manipular a execução de experimentos.

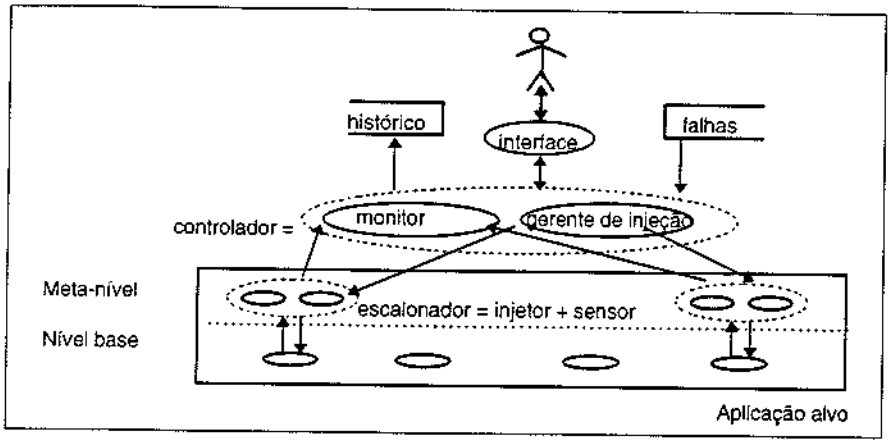


Figura 1: Uma arquitetura reflexiva para injeção de falhas e monitorização.

3.2 Injeção de falhas

Na ferramenta proposta, falhas são injetadas em tempo de execução, assim um mecanismo de ativação das falhas injetadas é necessário. A ativação de falhas pode ser temporal ou espacial [CMS95]. Na ativação temporal, falhas são ativadas quando um tempo predefinido é atingido. Na ativação espacial, falhas podem ser ativadas quando a execução da aplicação passa por um endereço previamente especificado [KKA92], e/ou certos dados são utilizados [CMS95]. Os tipos de falhas considerados são uma outra característica importante de uma ferramenta de injeção de falhas por software. Ambos os aspectos são discutidos a seguir.

3.2.1 Mecanismo de ativação

Em nossa abordagem a ativação das falhas é espacial, utilizando o mecanismo de interceptação de mensagens OpenC++1.2. A intercepação de mensagens é a característica provida pelo protocolo de metaobjeto da linguagem OpenC++1.2 para controlar objetos reflexivos, e pode ocorrer na chamada de métodos reflexivos ou no acesso a atributos reflexivos.

A figura 2 mostra como falhas são ativadas pela chamada de um método reflexivo. Quando um método reflexivo é chamado no nível base, a chamada é capturada e manipulada no meta-nível (por um *escalonador*) através do método *Meta_MethodCall* (①). Este meta-método verifica

se a falha dever ser injetada. Em caso afirmativo invoca um *injetor* para injetar a falha especificada. Então o *escalonador* invoca o método da aplicação usando outro método *Meta_HandleMethodCall* (②③). No final do *Meta_MethodCall*, o controle retorna para o nível base e os resultados são retornados ao chamador na aplicação como em uma chamada de método normal (④). O processo é o mesmo para o acesso a atributos reflexivos.

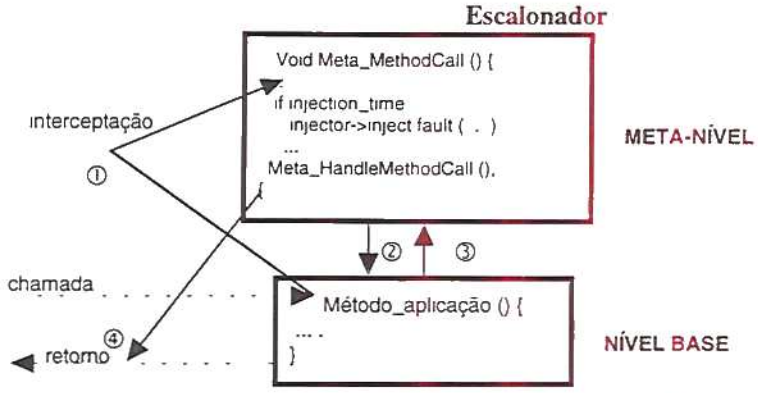


Figura 2: Ativação da injeção via chamada de método reflexivo.

Como um objeto pode ser injetado e/ou monitorado pelo metaobjeto, o *escalonador* precisa de um nome-de-categoria (ver item 2.2) para identificar a ação a ser executada de acordo com o método ou atributo em questão.

3.2.2 Tipos de falhas

Em nossa abordagem de teste, fazemos um paralelo com circuitos de hardware considerando objetos como circuitos de software, como proposto em [Hof89]. Quando consideramos objetos como circuitos, nós podemos usar a mesma correspondência utilizada por Hoffman; ou seja, cada método (ou atributo) público pode corresponder a um pino de circuito, ou a um conjunto relacionado de pinos. Fazendo um paralelo com a injeção de falhas por hardware, podemos injetar falhas internas, afetando as características privadas[2] do objeto, ou falhas externas, afetando a interface do objeto.

Uma outra dimensão da classificação das falhas é baseada no seu padrão de repetição: falhas podem ser transientes (nunca repetem), intermitentes (repetem conforme um intervalo predefinido) ou permanentes (sempre repetem). Nossa ferramenta pode injetar esses três tipos de falhas.

4. Alguns resultados

A fim de validar nossa ferramenta foi realizada uma série de experimentos tendo como alvo a aplicação orientada a objetos tolerante a falhas, Pilha Robusta [Pra98]. Essa aplicação usa dois mecanismos alternativos para tolerar falhas de software: blocos de recuperação e N-versões. Três variantes de pilha com diversidade de projeto são utilizadas.

As falhas consideradas nos experimentos foram baseadas na corrupção de valores dos parâmetros de métodos públicos; quanto ao padrão de repetição foram injetadas falhas transientes, intermitentes e permanentes.

Os testes realizados serviram para revelar erros de implementação da aplicação alvo e verificar diferenças no potencial de detecção de erros dos tipos de falhas considerados pela ferramenta.

---

[2] Embora Openc++ 1.2 não permita reflexão em caracterísiticas privadas de um objeto, falhas internas podem ser consideradas para as características protegidas.41

Foram revelados dois erros: um erro na implementação de uma das variantes e um erro na implementação do mecanismo blocos de recuperação. Quanto a eficiência na detecção de erros, nos experimentos realizados verificamos que:

- o momento da injeção de falhas foi um fator muito importante: as falhas injetadas desde o início da execução da aplicação alvo não revelaram o erro de implementação no mecanismo de blocos de recuperação, apenas quando começamos a variar o início da ativação da injeção de falhas esse erro foi descoberto;
- as falhas transientes e intermitentes foram mais eficientes: as falhas permanentes não detectaram o erro de implementação de uma das variantes.

5. Conclusões

O uso da abordagem metaobjeto permite implementar as características de injeção de falhas e monitorização independentemente da funcionalidade da aplicação. Comparando-se a outras abordagens de injeção de falhas por software duas outras vantagens podem ser destacadas. Primeiro, esta abordagem facilita a introdução da instrumentação no código da aplicação: não são inseridas explicitamente instruções de *trap* no código da aplicação e não é necessário executar a aplicação em modo traço. Segundo, esta abordagem simplifica o uso, pois não é preciso executar a aplicação em modo privilegiado, nem é necessário utilizar as características especiais do hardware as quais não são disponíveis para a maioria dos usuários.

Os experimentos realizados com a ferramenta demonstraram sua eficiência na detecção de erros de implementação da aplicação alvo. A aplicação já havia sido avaliada utilizando teste de análise de mutantes, mas os problemas de implementação não haviam sido revelados.

Nosso próximo passo é tentar realizar testes em um aplicação orientada a objetos reflexiva. Neste caso, os mecanismos de tolerância a falhas são implementados no meta-nível e as características de injeção e monitorização deverão ser incorporadas a um meta-meta-nível.

Referências

[CMS95] Carreira, J.; Madeira, H.; Silva, J.G. "Xception: a software fault injection and monitoring in processor functional units". *5th IFIP International Working Conference on Dependable Computing for Critical Applications*, Urbana-Champaign, Illinois, USA, 1995, pp. 135 -149

[Chi93] Chiba S., *"Open-C++ Release 1.2 Programmer's Guide"*, Technical Report n° 93-3, Dept. Information Science, University of Tokyo, 1993.

[Hof89] Hoffman D., "Hardware Testing and Software IC's", *Proc. Pacific NW Software Quality Conference*. Portland, Oregon, sept 1989.

[HTI97] Hsueh, M.C.; Tsai, T.K.; Iyer, R.K."Fault Injection Techniques and Tools". *IEEE Computer*,April/1997, pp.75-82.

[KKA92] Kanawati, N.; Kanawati, G.; Abraham, J. "FERRARI: A Tool for the Validation of System Dependability Properties". *Proc. FTCS-22*, IEEE CS Press, Los Alamitos, Calif., 1992, pp. 336-344.

[Lis97] Lisboa, M.L.B. *"Computational Reflection in the Object Model"*. Tutorial presented at II Brazilian Symposium of Programming Languages, Campinas, SP, Brazil, 1997, 53 pages.

[Mae87] P. Maes, "Concepts and Experiments in Computaional Reflection". *Proc. OOPSLA'87*, p. 147-155, 1987.

[Pra98] D. Piubeli Prado. *"Implementação de Sistemas Tolerantes a Falhas Usando Programaçao Orientada a Objetos"*. Dissertação de Mestrado, Instituto de Computação, Universidade Estadual de Campinas, jan/98, 88 páginas.

Injeção de falhas em protocolos tolerantes a falhas utilizando a arquitetura Ferry clip

Marcos Renato Rodrigues Araujo
maraujo@dcc.unicamp.br

Eliane Martins
eliane@dcc.unicamp.br

Instituto de Computação – Universidade Estadual de Campinas
Caixa Postal 6065 – 13081.970 Campinas/SP

Resumo

Na validação de protocolos tolerantes a falhas são necessários testes que validem seus mecanismos de tolerância a falhas. Para isto um dos métodos utilizados é a injeção de falhas via software, dada sua simplicidade e baixo custo de desenvolvimento. Estes testes são chamados de testes de tolerância à falhas neste contexto.

Este artigo descreve uma ferramenta que está sendo desenvolvida para permitir realizar testes de tolerância à falhas de implementações de protocolos de comunicação. Esta ferramenta utiliza uma variação da arquitetura Ferry clip, modificada com o propósito de permitir a realização de testes de tolerância à falhas.

Abstract

For the validation of fault-tolerant protocols, the validation of their fault-tolerance mechanisms is needed. One method used with this aim is software implemented fault injection, since it is simple to use and has low development cost. In our context, this kind of testing is called fault-tolerance testing.

This paper describes a tool that is being developed to support fault-tolerance tests for communications protocol implementations. This tool extends the Ferry clip architecture to support fault-tolerance testing.

1 Introdução

Durante o desenvolvimento de um sistema podemos enumerar três estágios: projeto, implementação e validação. A fase de *projeto* consiste em estruturar o protocolo através dos requisitos, onde estão descritas todas as regras que o mesmo deve obedecer. A *implementação* consiste em traduzir o projeto elaborado na fase de projeto em termos computacionais, adaptando-se o mesmo às condições do ambiente de desenvolvimento. A *validação* consiste em assegurar que a implementação do protocolo esteja de acordo com a especificação do mesmo. A arquitetura *Ferry clip* [CLP+89] foi desenvolvida com a finalidade de dar suporte à validação de protocolos.

No desenvolvimento de um protocolo Tolerante a Falhas, deseja-se validar também os mecanismos de tolerância a falhas desenvolvidos. Uma das maneiras de se validar estes mecanismos é através da injeção de falhas, que consiste em inserir de maneira controlada falhas durante a execução do protocolo, observando o comportamento do mesmo quando submetido a elas. Injetores de falhas por software emulam falhas de hardware sem a necessidade de se desenvolver um hardware adicional, com conseqüente redução nos custos de desenvolvimento.

Este resumo descreve brevemente uma ferramenta atualmente em desenvolvimento que acrescenta à arquitetura Ferry clip um injetor de falhas via software. Esta arquitetura foi escolhida devido à sua portabilidade: sua estrutura altamente modular permite que ela possa ser usada em diferentes plataformas requerendo para isso poucas modificações.

2 O que é *Ferry clip*?

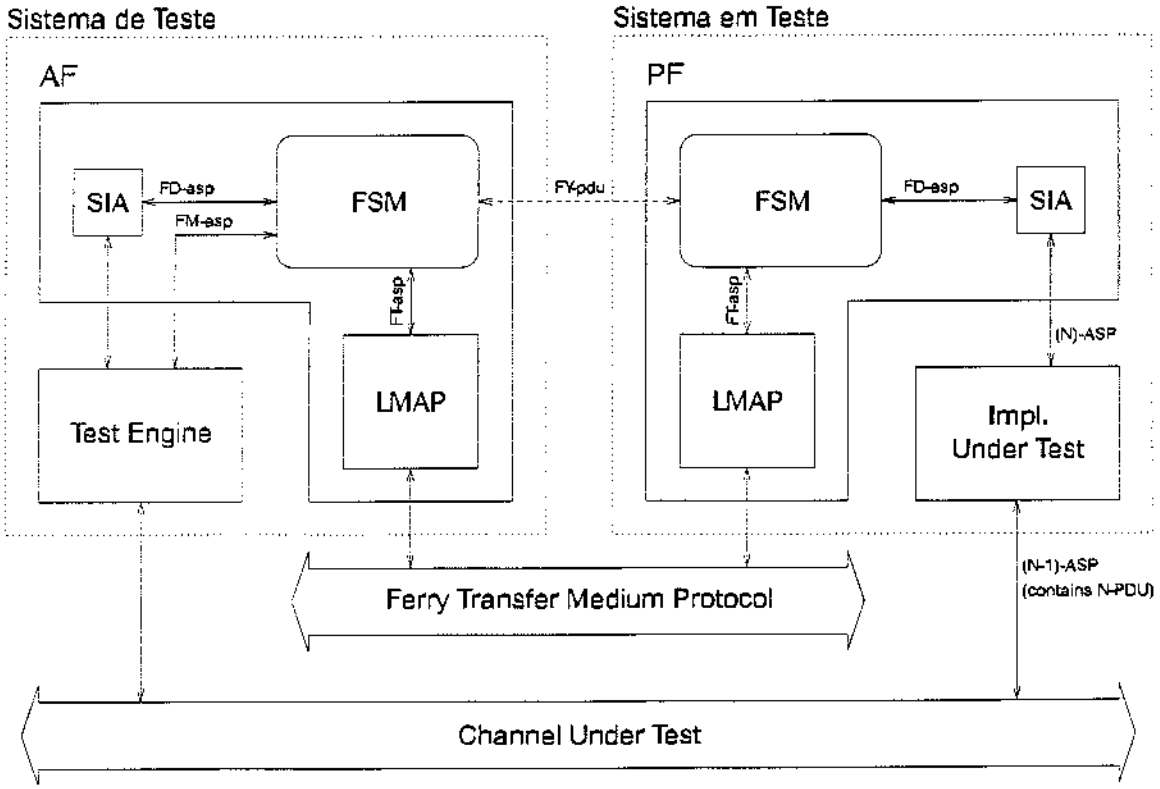


Figura 1 – Arquitetura Ferry clip

O conceito de Ferry clip [CLP+89] foi desenvolvido com o objetivo de dar suporte aos métodos de teste definidos pela ISO [ISO]. Basicamente consiste em transportar os dados de teste de maneira transparente do Sistema de Teste (responsável pela coordenação dos testes) ao Sistema em Teste (responsável pela execução dos mesmos), permitindo que ambos os testadores residam

junto ao Sistema de Teste, simplificando a sincronização entre UT e LT, minimizando assim a quantidade de software que reside junto ao Sistema em Teste.

Um sistema Ferry clip (figura 1 [CVD92]) consiste de dois componentes: o Active Ferry clip (AF), residente no Sistema de Teste, e o Passive Ferry clip (PF), residente no Sistema em Teste. Ambos os Ferry's trocam dados através de um protocolo de comunicação denominado Ferry Control Protocol (FCP), que utiliza os serviços de um canal de comunicação confiável denominado Ferry Transfer Medium Protocol (FTMP).

O Test Engine agrupa as funções do Testador Superior e o Testador Inferior, sendo também responsável pelo envio das instruções à FSM do AF tal como estabelecimento de conexão com PF, desconexão e outros dados de controle. As informações circulam entre AF e PF através de ASP's (Abstract Service Primitives): FD-asp para transporte de dados entre Test Engine e IUT, FM-asp para envio de comandos às FSM's e FT-asp, para o transporte entre AF e PF. Ambas as FSM's trocam dados através de FY-PDU's (Ferry Protocol Data Units), encapsuladas em FT-asp's. Cada Ferry possui três sub-módulos: FSM (*Finite State Machine*), que reúne as funções de gerenciamento interno de cada Ferry; SIA (*Service Interface Adapter*), encarregado de interagir com a IUT e transportar através do Ferry, de maneira transparente, sua interface ao Test Engine; e LMAP (*Lower Mapping Module*), responsável pelo transporte de dados entre os Ferry's. É esta modularidade que facilita a portabilidade entre sistemas distintos

3 Injeção de falhas

A injeção de falhas é um método de validação de protocolos que possuem métodos de tolerância e recuperação de falhas. Durante a validação são introduzidas falhas no sistema com o objetivo de estimular estes mecanismos. Os principais objetivos são [SW97]:

- Validação dos procedimentos de verificação;
- Validação dos mecanismos de tolerância a falhas, permitindo:
  1. previsão de falhas;
  2. eliminação de falhas;

A injeção de falhas pode ser feita basicamente de três maneiras: *injeção física* (via hardware), *simulação lógica* e *injeção lógica* (via software).

A *injeção física* ou via hardware consiste em injetar falhas diretamente no hardware da implementação, tais como sinais elétricos ou distúrbios eletromagnéticos. Estas técnicas em geral possuem alto custo de implementação devido à necessidade de desenvolvimento de hardware próprio com esta finalidade, além de eventualmente causar danos físicos à implementação.

A *simulação lógica* consiste em desenvolver uma representação da IUT em software e estudar seu comportamento. Esta técnica está associada a um alto custo de desenvolvimento devido à necessidade de se representar toda a implementação em software, algo nem sempre possível, além de não haver garantias de que a simulação corresponderá a uma implementação real.

A *injeção lógica* ou *injeção via software* possui duas características básicas: tem baixo custo de implementação e não sujeita a IUT a danos físicos, e consiste em um meio-termo entre injeção via hardware e simulação de falhas: emula falhas de hardware no software da própria implementação, de modo que a IUT detecte a simulação como uma falha; deste modo não há necessidade de um hardware dedicado à injeção de falhas, nem necessidade de uma representação da IUT em software. É um método muito mais simples de ser implementado e a um custo muito mais baixo.

Sua principal desvantagem é a limitação para representar determinados tipos de falhas (controle da CPU por exemplo) além de afetar as características de temporização do sistema, limitando seu uso em sistemas de tempo real. Tais desvantagens podem ser minimizadas através da redução da quantidade de software associado à injeção de falhas. A arquitetura Ferry clip foi escolhida aqui com o objetivo de minimizar o software associado à implementação em teste.

4 A Ferramenta

Nossa ferramenta pretende dar suporte ao teste de protocolos visando um alto grau de portabilidade e minimização do Sistema em Teste através da utilização da arquitetura Ferry clip, dando suporte a testes de protocolos tolerantes à falhas, com ou sem injeção de falhas. Neste caso o requisito mínimo de nossa ferramenta é ter acesso às interfaces superior e inferior da IUT.

4.1 Características

A arquitetura Ferry clip permite um alto grau de minimização do código residente junto à implementação e um alto grau de portabilidade para a ferramenta, uma vez que para uma diferente plataforma apenas alguns módulos devem ser adaptados. No caso de testes realizados em IUT's distintas, apenas os módulos AF-SIA e PF-SIA necessitam modificação. No nosso esquema o injetor de falhas é colocado na camada abaixo da IUT.

Utiliza-se uma ferramenta para geração automática dos testes. Esta ferramenta está descrita em [SM98] e gera casos de teste determinísticos a serem usados pelo UT e o LT. A geração automática de falhas ainda não foi implementada.

4.2 Modelo esquemático

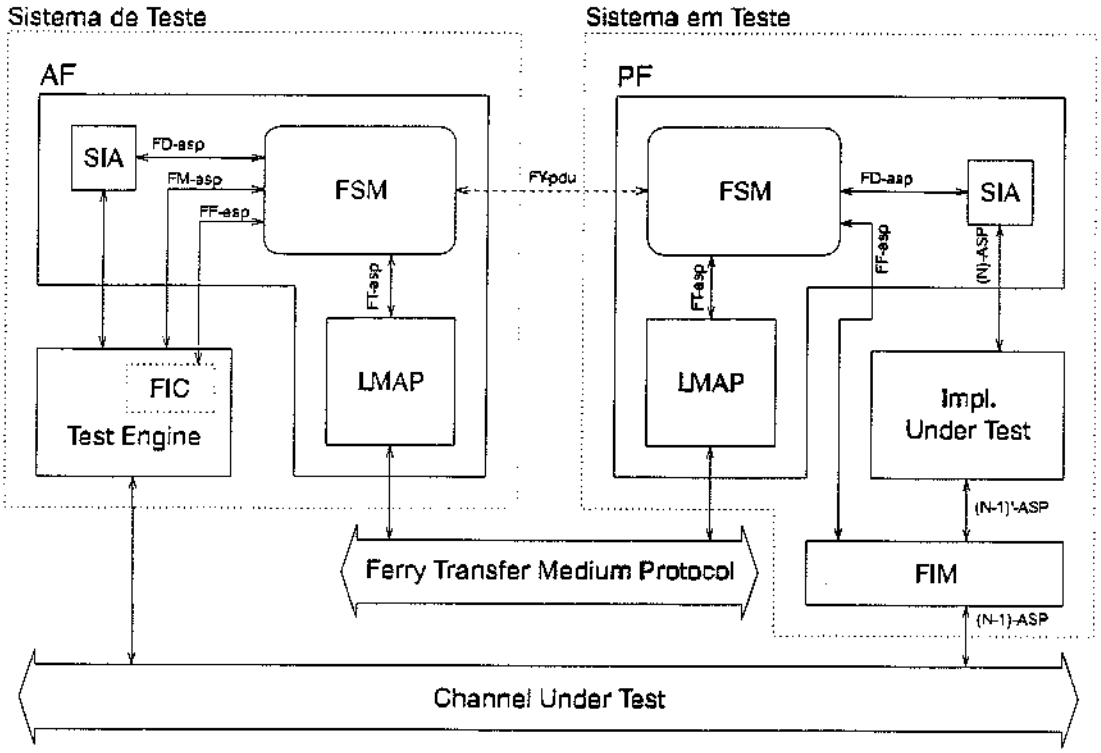


Figura 2 – Modelo esquemático da ferramenta

A Figura 2 mostra o modelo esquemático da ferramenta. Percebe-se pequenas alterações ao modelo original do Ferry clip. De fato, a idéia é ser o mais próximo desta arquitetura possível. A principal diferença esquemática está no acréscimo do FIC ao Test Engine e do FIM ao Sistema em Teste. O FIC (Fault Injection Controller) é responsável pelo gerenciamento remoto do FIM (Fault

Injection Module). O envio de dados ao FIM é feito através de FF-asp's, acrescidas ao modelo original do Ferry clip (ver Figura 1).

O FIM, como mencionado, é o módulo responsável pela interceptação e tratamento das mensagens que passam pela camada inferior da IUT. Em um caso típico, o FIC envia ao FIM as instruções sobre como ele deve proceder. A figura 3 ilustra o diagrama esquemático do FIM.

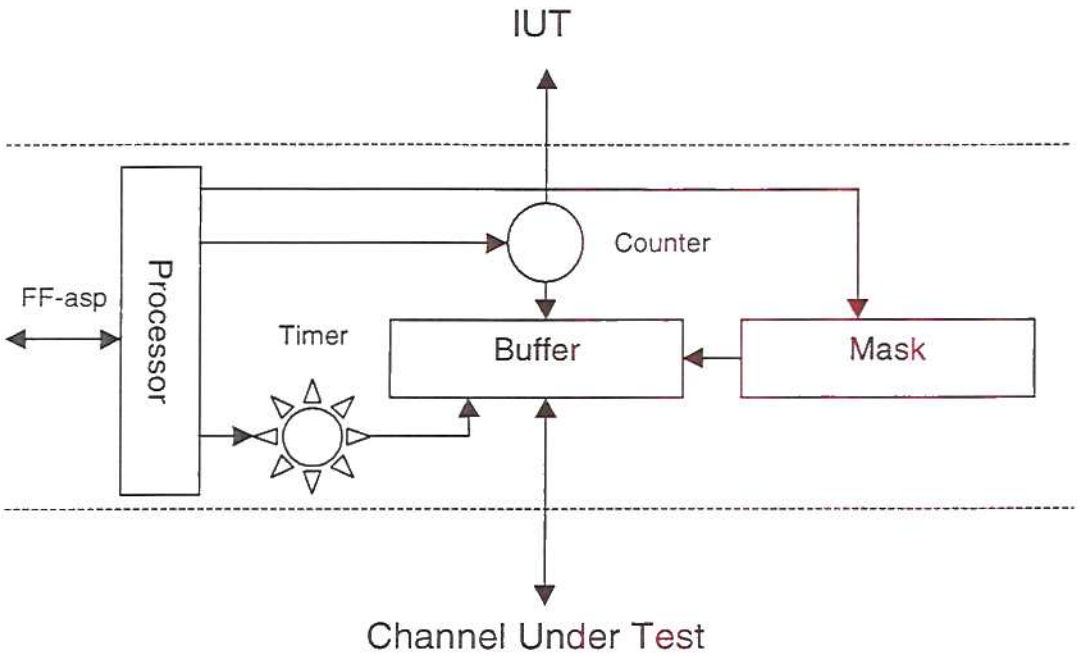


Figura 3 – Fault Injection Module

O FIM injeta falhas diretamente nas mensagens que circulam através dele, sendo responsável por três atividades: filtragem (interceptação de mensagens), tratamento (injeção da falha em si) e injeção (devolver a mensagem, alterada ou não, ao sistema). O FIM é capaz de gerar quatro tipos de falhas de comunicação:

1. *duplicação*: a mensagem é enviada várias vezes;
2. *supressão*: a mensagem é suprimida totalmente;
3. *corrupção*: a mensagem é corrompida de algum modo;
4. *temporização*: a mensagem é enviada com atraso;

O módulo *processor* é responsável pela recepção das instruções e pela sua execução sobre os outros módulos. O *buffer* armazena a última mensagem; *mask* armazena a máscara para a injeção das falhas e o *timer* armazena a informação para atraso da mensagem.

Para a duplicação de mensagens, o *buffer* armazena a mensagem e o *processor* informa quantas vezes a informação deve ser repetida antes de ser eliminada; o *timer* controla as falhas de temporização informando quanto o *buffer* deve atrasar antes de enviar o pacote (normalmente o *timer* fica em zero, enviando a mensagem imediatamente); o *mask* armazena a máscara de corrupção das mensagens. O *counter* é utilizado para contagem de mensagens que passam pelo FIM, necessário para saber qual será a mensagem na qual será aplicada a falha; neste esquema o

usuário da ferramenta informa que a mensagem na qual será injetada a falha será a n-ésima mensagem a passar pelo FIC.

Como a ferramenta pode ser utilizada para testes de conformidade seja com ou sem injeção de falhas, um esquema de script compatível deve ser considerado. Escolheu-se trabalhar com um único script responsável pela programação do FIM e pelos casos de teste. Para tanto o script é dividido em duas partes: a primeira prepara o FIM para a injeção de falhas, aguardando confirmação dele para cada instrução enviada; a segunda consiste do script com os casos de teste. Uma vez preparado, o injetor fica ativo até o fim dos casos de teste. O envio das instruções e a espera da confirmação é feita pelo FIC.

4.3 Implementação

A implementação da ferramenta está sendo feita com o objetivo inicial de se testar uma implementação do protocolo X.25 desenvolvido para plataforma PC. Para tanto foi escolhido o sistema operacional Windows95 para dar suporte ao Sistema em Teste. Visando testar a funcionalidade da ferramenta em plataformas distintas, o Sistema de Teste está sendo implementado em plataforma Solaris 2.5, ambas as partes desenvolvidas em C++ com Orientação a Objetos. Para troca de dados entre as partes no Sistema de Teste utiliza-se o esquema de pipes, sob a forma de sockets Unix. Este esquema permite que um único Test Engine "dispare" diversos processos AF simultaneamente e mantenha conexão com todos eles sem necessidade de modificações na ferramenta.

Atualmente estão concluídas em ambiente Solaris as implementações do AF e do PF. Para efeito de validação foram feitos dois testes básicos sobre estes componentes: interconexão e troca de dados. Para o teste de interconexão simplesmente colocou-se o AF e o PF executando em máquinas distintas e, através de uma interface simples, solicitou-se uma conexão e uma desconexão. Ambos os comandos foram aceitos sem problemas.

Para o teste de troca de dados desejava-se também verificar o desempenho do canal de comunicação. Para tanto, utilizou-se o mesmo esquema de AF e PF, este executando em modo *'loopback'* (retornar qualquer dado que chega pelo mesmo caminho), em máquinas distintas e uma interface simples com o AF cuja finalidade é solicitar abertura de conexão com um determinado PF, enviar uma determinada informação 10 mil vezes, comparar o dado recebido com o que foi enviado e desconectar. O tempo de execução deste esquema foi cronometrado dez vezes, obtendo-se dez medidas de tempo distintas.

As medidas obtidas foram (em segundos, em ordem crescente): 14.201s, 14.592s, 15.245s, 15.884s, 17.294s, 17.818s, 19.674s, 19.958s, 24.167s, 24.201s, dando uma média de 18.303s. Desprezando-se o tempo gasto na conexão e na desconexão, dividiu-se o tempo gasto pelo número de mensagens trocadas (10 mil) e obteve-se o tempo médio de ida e volta de uma mensagem cujo valor é 1.83ms, resultando em 0.92ms o tempo gasto médio para uma mensagem ser enviada de uma máquina a outra. O protocolo de comunicação utilizado foi o TCP/IP, garantindo-se assim a não ocorrência de mensagens perdidas. Ambas as máquinas localizam-se na mesma rede local.

5 Conclusão

Foi descrita aqui uma nova ferramenta que está sendo implementada para o teste de protocolos e seus mecanismos de tolerância à falhas. Para tanto utiliza-se o conceito de injeção de falhas por software para validação dos mecanismos de tolerância à falhas desenvolvidos para o protocolo, e o conceito de Ferry clip como arquitetura básica para desenvolvimento da ferramenta.

Atualmente a ferramenta está em fase de implementação, estando prontos o AF e PF, ambos implementados em Solaris, e uma versão beta do Test Engine, utilizada para testar a conexão entre AF e PF. O próximo passo é portar o PF para o ambiente Windows95, concluir a implementação do FIM e concluir o Test Engine.

Apêndice: Nomenclatura utilizada

AF: Active Ferry

ASP: Abstract Service Primitives

FIC. Fault Injector Controller

FIM: Fault Injection Module

FSM: Finite State Machine

FD-asp: Ferry Data Abstract Service Primitives

FM-asp: Ferry Management Abstract Service Primitives

FF-asp: Ferry Fault Abstract Service Primitives

FT-asp: Ferry Transport Abstract Service Primitives

IUT: Implementation Under Test

LMAP: Lower-Mapping Module

PF: Passive Ferry

SIA: Service Interface Adapter

Bibliografia

[CLP+89] Chanson, S. T., Lee, B. P., Parakh, N. J., Zeng, H. X., *"Design and Implementation of a Ferry clip Test System"*, Proc. 9th IFIP Symposium on Protocol Specification Testing & Verification, Enschede, Holanda (Junho 1989)

[CVD92] Chanson, S. T., Vuong, S., Dany, H., *"Multi-party and interoperability testing using the Ferry clip approach"*, Computer Communications, vol 15, nº 3 (Abril 1992)

[ISO] ISO TC97/SC21, DIS 9646, *"OSI Conformance Testing Methodology and Framework"*, ISO, Genebra, Suíca (Novembro 1989)

[SM98] Sabião, S. B., Martins, E., *"CONDADO – uma ferramenta para geração de testes de protocolos combinando Controle e Dados"*, submetido ao 16º Simpósio Brasileiro de Redes de Computadores (Janeiro 1998)

[SW97] Sotoma, I., Weber, T. S., *"AFIDS - Arquitetura para Injeção de Falhas em Sistemas Distribuídos"*, Anais do XV Simpósio Brasileiro de Redes de Computadores, São Carlos/SP (1997)

Um Algoritmo para Diagnóstico de Redes de Topologia Arbitrária

Elias Procópio Duarte Jr.
Universidade Federal do Paraná, Depto. de Informática
Caixa Postal 19081 Curitiba 81531-990 PR Brasil
e-mail: elias@inf.ufpr.br

Resumo

É crescente a demanda por sistemas de gerência de redes capazes de diagnóstico de falhas e problemas de desempenho. É importante que tais sistemas sejam, eles próprios, tolerantes a falhas. Neste trabalho, apresentamos um algoritmo para diagnóstico de falhas em redes de topologia arbitrária, aplicável a sistemas integrados de gerência. O algoritmo permite o diagnóstico de falhas nos canais de comunicação da rede, e o cálculo da conectividade sob o ponto de vista de qualquer nodo sem falhas. Trata-se de uma abordagem tolerante a falhas pois, como o algoritmo é totalmente distribuído, mesmo que ocorram falhas na rede, os nodos sem falha continúam monitorando a rede continuamente.

Abstract

There is a growing demand for network management systems that are capable of effectively diagnosing faults and performance problems. To achieve this goal, it is important that those systems themselves be fault-tolerant. In this work we present a system-level diagnosis algorithm for general topology networks, that can be applied for network fault management. The algorithm allows link fault diagnosis, after which nodes compute network connectivity. It employs the minimum number of tests, i.e. one per link per testing interval. The latency of the algorithm is proportional to the diameter of the graph corresponding to the network. This approach is fault-tolerant in the sense that no matter which portion of the network is faulty, the fault-free nodes keep on monitoring the network continuously.

1 Introdução

A constante expansão das redes de computadores dentro de empresas e organizações implica num aumento dos custos associados a eventuais falhas e problemas de performance. Ao mesmo tempo, as redes são cada vez complexas, no sentido de que se constituem de componentes heterogêneos, produzidos por uma variedade de fabricantes. Assim, é fundamental que o gerente da rede tenha a sua disposição um conjunto de ferramentas que lhe permitam evitar problemas, e resolvê-los com rapidez se por ventura ocorrerem.

A maioria dos sistemas de gerência de rede atuais se baseia no protocolo SNMP ("Simple Network Management Protocol") [3]. Nestes sistemas, uma máquina é responsável por supervisionar toda a rede. Esta máquina é chamada *NMS* ("Network Management Station"), e em geral apresenta uma interface gráfica através da qual o gerente humano tem acesso ao sistema. Para monitorar a rede, o NMS se comunica com uma série de *agentes*, cada um deles responsável por monitorar um componente da rede, que pode ser uma máquina, um canal de comunicação, um hub, um protocolo, entre outros. A comunicação entre NMS e agentes se dá através de um protocolo de gerência de redes, como o SNMP. O NMS pode realizar um polling periódico dos agentes, ou então receber "traps", quer dizer, interrupções que informam situações de emergência.

Os sistemas centralizados apresentam dois problemas: em primeiro lugar, se o NMS falhar, então a rede deixará de ser monitorada. Além disso, outro problema é a concentração de mensagens de gerência em um único nodo, o que pode acarretar efeitos negativos na sua performance. Neste trabalho descrevemos um algoritmo totalmente distribuído e tolerante a falhas para monitorar uma rede de topologia arbitrária.

Esta estratégia é baseada na teoria de diagnóstico de falhas a nível de sistema ("system-level diagnosis"). Esta teoria já vem se desenvolvendo há mais de 30 anos. Os algoritmos práticos para diagnóstico a nível de sistema podem ser divididos em duas categorias: os que assumem que entre todo par de nodos na rede existe um único canal de comunicação, quer dizer, que o grafo do sistema é completo, e os que permitem que a rede tenha topologia arbitrária. O Hi-ADSD é um algoritmo recente para diagnóstico de redes totalmente conexas [1].

O algoritmo NBND ("Non-Broadcast Network Diagnosis") descrito neste trabalho foi introduzido inicialmente em [2]. Ele permite o diagnóstico de falhas nos canais de comunicação da rede e o cálculo da conectividade sob o ponto de vista de qualquer nodo sem falhas. Trata-se de uma abordagem tolerante a falhas pois, como o algoritmo é totalmente distribuído, mesmo que ocorram falhas na rede, os nodos sem falha continúam monitorando a rede continuamente. O algoritmo emprega o menor número de testes possível, e apresenta latência igual ao diâmetro do grafo que representa a rede.

O resto do trabalho está estruturado da seguinte forma. Na seção 2, é feita uma revisão de algoritmos para diagnóstico em redes de topologia arbitrária. Na seção 3 o algoritmo NBND é descrito, junto com um exemplo de execução. A seção 4 conclui o trabalho.

2 Algoritmos para Diagnóstico a Nível de Sistema

Considere um sistema que consiste de N unidades, ou nodos, que podem estar *falhos* ou *sem-falha*. O objetivo dos algoritmos de diagnóstico a nível de sistema é que todos os nodos sem-falha determinem o estado de todos os nodos do sistema. Podem ocorrer dois tipos de *evento* num sistema: um nodo sem-falha se tornar falho, e um nodo falho se tornar sem-falha. Cada nodo executa testes em outros nodos, e os nodos sem-falha são capazes de determinar corretamente o estado dos nodos testados por eles.

Se pensarmos em cada teste como uma aresta direcionada do testador para o nodo testado, então o conjunto de todos os nodos e todos os testes é o chamado "grafo de

testes".

Os algoritmos para diagnóstico são executados em etapas, ou *rounds*. Para entender o conceito de etapa, é necessário entender antes o conceito de intervalo de testes. Um nodo executa testes a cada intervalo de testes, que pode ser, por exemplo, 10 segundos. Quando todos os nodos sem-falha tiverem executado seus testes, uma etapa se completou. O número de etapas necessárias para que todos os nodos sem-falha do sistema façam o diagnóstico de um determinado evento é uma medida importante da eficiência de um algoritmo de diagnóstico, e é chamada *latência* do algoritmo.

2.1 Algoritmos para Redes de Topologia Arbitrária

Existem duas categorias principais de algoritmos de diagnóstico. Uma categoria realiza diagnóstico em redes nas quais existe um canal de comunicação direto entre quaisquer dois nodos da rede. A outra categoria admite que a topologia seja arbitrária, quer dizer, entre dois nodos do sistema pode haver ou não um canal de comunicação direto. No caso de não haver, para que os dois nodos se comuniquem, eles devem usar nodos intermediários.

Em [4], Bagchi e Hakimi introduziram um algoritmo para diagnóstico em redes de topologia arbitrária que utiliza o menor número possível de mensagens. Os nodos sem-falha formam um grafo de testes, através do qual as mensagens de diagnóstico são transmitidas. Entretanto, este algoritmo é executado off-line: ele não permite um monitoramento dinâmico e contínuo da rede.

Em [5] Bianchini e outros introduziram e simularam o algoritmo "Adapt". Este algoritmo é executado on-line: quando ocorre um evento, os nodos sem-falha se reorganizam de forma a manter o grafo de testes conexo, se possível. Este grafo de testes é o menor grafo fortemente conexo cujos nodos são os nodos sem falha, e cujas arestas são os canais de comunicação entre tais nodos. Para construir o grafo de testes, o algoritmo emprega um procedimento distribuído que requer grandes quantidades de mensagens enormes.

Recentemente, Rangarajan e outros introduziram um novo algoritmo para diagnóstico em redes de topologia arbitrária [6]. Este algoritmo, que chamaremos de RDZ (das iniciais dos autores), é também executado on-line, como o algoritmo Adapt. Além disso, o grafo de testes garante o menor número de testes por nodo, quer dizer, cada nodo é testado por apenas um único testador. O algoritmo apresenta a melhor latência, decorrente de uma estratégia paralela de disseminação das mensagens de diagnóstico. Entretanto, certas configurações de falhas constituem eventos que são diagnosticados apenas *eventualmente*, quer dizer, não há uma garantia de quando estes eventos são diagnosticados. Este problema impede o uso do algoritmo RDZ para gerência de falhas de redes.

3 O Algoritmo NBND

Nesta seção descrevemos o algoritmo NBND ("Non-Broadcast Network Diagnosis"). Este algoritmo foi desenvolvido para ser aplicado a sistemas de gerência de falhas em redes. Trata-se de uma estratégia que permite o diagnóstico de time-outs de canais de comunicação, e calcula conectividade dos nodos da rede, usando o menor número de testes

possível: um por canal. Além disso, o algoritmo apresenta a melhor latência possível, proporcional ao diâmetro da rede.

O algoritmo NBND se baseia no fato de que é possível determinar se um determinado canal está dando "time-out" ou se está sem-falha. Entretanto, é impossível distinguir se um nodo está falho ou se todos os canais de comunicação para tal nodo é que estão falhos. Como mostra a figura 1, estas configurações de falhas são ambíguas. Assim, ao invés de calcular quais nodos estão falhos e quais nodos estão sem falhas, o algoritmo calcula quais nodos estão atingíveis e quais nodos estão estão inatingíveis. Resumindo, os estados de um canal de comunicação são *sem-falha* e *timed-out* e os estados de um nodo são *sem-falha* ou *inatingível.*

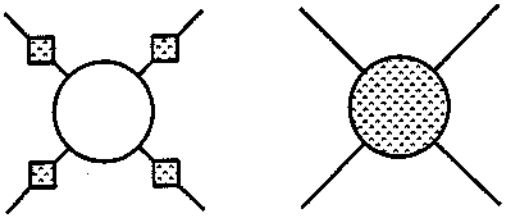


Figura 1: Configurações de falhas ambíguas.

O algoritmo utiliza o menor número possível de testes: cada canal de comunicação é testado por apenas um dos nodos por ele interligados. A estratégia proposta em [2] é a escolha do nodo com maior identificador como testador de um canal específico. No caso deste nodo, ou do próprio canal de comunicação, se tornar falho, o nodo de menor identificador passa então a executar testes. É importante observar que esta distribuição de testes pode causar situações em que alguns nodos executem diversos testes, enquanto outros executam poucos. Uma estratégia mais justa é os os nodos se alternarem como testadores e testados a cada intervalo.

Cada nodo mantém uma tabela com o estado de todos os nodos da rede. Esta tabela é inicializada em zero, e incrementada de uma unidade cada vez que o nodo sofrer um evento. Assim, valores pares indicam um nodo sem falhas, e valores ímpares indicam nodos falhos.

Quando um novo evento é descoberto, o nodo testador dissemina a informacao para todos seus vizinhos em paralelo, e estes, por sua vez, repetem o processo. Para reduzir o número total de mensagens, estas contêm um campo no qual os identificadores de nodos que já sabem do novo evento são armazenados. As mensagens recebidas podem ser opcionalmente processadas apenas uma vez a cada intervalo de testes. Neste caso, elas devem ser armazenadas em uma tabela, na medida em que são recebidas.

Após receber informação sobre um novo evento, cada nodo executa um algoritmo para determinar a conectividade da rede, quer dizer, se esta está particionada ou não, e em que pontos. Em resumo, o algoritmo é como apresentamos a seguir:

```
BEGIN
 /* at nodo i */
 DO FOREVER
  FOR each link i-j, that connects node i to node j
    IF my_turn(j) /* computes based on last message received from j */
    THEN test link i-j;
         IF link i-j is fault-free
         THEN i and j exchange messages;
         IF there is a new event
         THEN update link status; disseminate messages;
  compute node reachability;
  SLEEP(testing interval)
END;
```

3.1 Exemplo de Diagnóstico

Nesta seção apresentamos um exemplo da execução do algoritmo NBND para a rede da figura 2. Em cada aresta do grafo, os arcos estão direcionados do nodo testador para o nodo testado. Inicialmente todos os nodos estão livres de falhas.

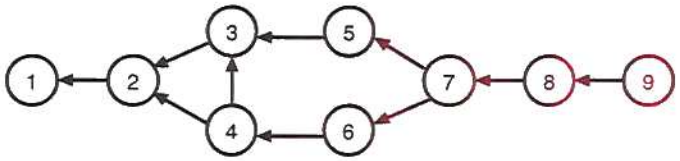


Figura 2: Grafo de testes da rede do exemplo.

Considere o seguinte evento nesta rede: o canal de comunicação entre o nodo 3 e o nodo 5 fica falho no tempo t=100. No próximo intervalo de testes, o nodo 5 detecta o evento, é o instante de tempo t=120. Ele dissemina mensagens para seu vizinho sem-falha, o nodo 7. Este processa a informacao em t=150, e dissemina para seus vizinhos, os nodos 6 e 8. Neste intervalo, como o nodo 3 não recebeu informações sobre o canal que o liga ao nodo 5, ele o testa, e descobre o evento. A informacao é disseminada para os nodos 2 e 4. O nodo 4 recebe mensagem tambem do nodo 6, referente ao mesmo evento. Neste momento, o nodo 9 recebe mensagem de diagnóstico do nodo 8. Os nodos 4 e 9 nao disseminam mensagens, pois sabem que tanto o nodo 8, como 2 e 6 já sabem do evento. Em t=180, o nodo 1 recebe a mensagem do nodo 2, completando o diagnóstico. São necessários, ao todo, 3 intervalos de testes para o diagnóstico completo. São disseminadas 8 mensagens sobre o evento. Apenas o nodo 6 recebe duas mensagens sobre o mesmo evento.

4 Conclusões

Neste artigo apresentamos um algoritmo para diagnóstico de redes de topologia arbitrária. O propósito do algoritmo é ser aplicado para sistemas de gerência de redes de computadores. Além de apresentar a melhor latência possível, proporcional ao diâmetro da rede, o algoritmo emprega o menor número de testes, um por canal de comunicação por intervalo de testes. Ao descobrir um novo evento, cada novo dissemina mensagens de diagnóstico em paralelo para seus vizinhos. São previstos mecanismos para evitar certas mensagens redundantes. Espera-se para breve resultados de simulações do algoritmo sobre redes de diversas tecnologias. Além disso, o algoritmo está sendo implementado integrado a uma sistema de gerência de redes baseado em SNMP.

Referências

[1] E.P. Duarte Jr., and T. Nanya, "A Hierarchical Adaptive Distributed System-Level Diagnosis Algorithm," *IEEE Transactions on Computers*, pp.34-45, Vol.47, No.1, Jan 1998.

[2] E.P. Duarte Jr., G. Mansfield, T. Nanya, and S. Noguchi, "Non-Broadcast Network Fault Monitoring Based on System-Level Diagnosis," *Proc. IEEE/IFIP IM'97*, pp.597-609, San Diego, May 1997.

[3] M.T. Rose, *The Simple Book - An Introduction to Internet Management*, 2^{nd} ed., Prentice-Hall, Englewood Cliffs, NJ, 1994.

[4] A. Bagchi, and S.L. Hakimi, "An Optimal Algorithm for Distributed System-Level Diagnosis," *Proc.* 21^{st} *Fault Tolerant Computing Symp.*, June, 1991.

[5] M. Stahl, R. Buskens, and R. Bianchini, "Simulation of the Adapt On-Line Diagnosis Algorithm for General Topology Networks," *Proc. IEEE* 11^{th} *Symp. Reliable Distributed Systems*, October 1992.

[6] S.Rangarajan, A.T. Dahbura, and E.A. Ziegler, "A Distributed System-Level Diagnosis Algorithm for Arbitrary Network Topologies," *IEEE Transactions on Computers*, Vol.44, pp. 312-333, 1995.

Comparação de Desempenho de Algoritmos de Recuperação Síncrono e Assíncrono

Sérgio Luis Cechin Ingrid Jansch-Pôrto
{cechin, ingrid}@inf.ufrgs.br
Curso de Pós-Graduação em Ciência da Computação
Instituto de Informática - UFRGS
Caixa Postal 15064 - CEP 91501-970
Porto Alegre - RS - Brasil

Resumo

A recuperação de processos por retorno pode ser implementada seguindo paradigmas síncrono ou assíncrono. Pretende-se, neste artigo, apresentar alguns resultados teóricos da comparação de desempenho entre dois algoritmos das categorias citadas, tomando-se por base os algoritmos de Koo e Toueg (síncrono) e o de Juang e Venkatesan (assíncrono). O objetivo da comparação é demonstrar que as vantagens e desvantagens relativas dependerão das características das aplicações.

Abstract

In distributed systems, backward recovery has two main implementation paradigms: synchronous and asynchronous. In this paper, we intend to compare two representative algorithms on these groups and to present some theoretical results. Koo & Toueg synchronous algorithm and Juang & Venkatesan asynchronous one are considered. Our goal is to demonstrate that the advantages and disadvantages between them are related to the characteristics of the applications.

1 Introdução

A recuperação de processos em sistemas distribuídos apresenta algumas dificuldades devidas à características do próprio ambiente: o sistema é formado por um conjunto de processos separados fisicamente sem o compartilhamento de memória ou de relógio (*clock*). Estes processos trocam informação exclusivamente através de mensagens [JAL94]. Na literatura, existem diversos trabalhos dedicados à apresentação de algoritmos destinados a realização de recuperação de processos neste contexto de sistemas. O presente trabalho objetiva comparar, através de modelos teóricos desenvolvidos pelo próprio autor [CEC98], dois algoritmos representativos dos enfoques síncrono e assíncrono, de forma a comprovar vantagens e desvantagens conceitualmente comentadas na literatura como parte da apresentação destes algoritmos.

Os algoritmos são aplicados a sistemas cujo comportamento baseia-se no que segue: o canal de comunicação é ideal com *buffers* infinitos, livres de erros e que entregam as mensagens na mesma ordem de envio [JAL94]; as falhas são temporárias e os processos atuam em *fail-stop*; e é admitida a ocorrência de mensagens perdidas e mensagens órfãs.

Para fins de análise, o tempo total de cada mensagem, tm - desde o envio por um processo até o recebimento pelo processo destino - foi dividido em três etapas: o "empacotamento", t_{me}; tempo para percorrer o canal de comunicação, t_{mt}; e o "desempacotamento", t_{md}.

2 Características do algoritmo síncrono

O algoritmo de Koo e Toueg [KOO87] baseia-se na determinação de pontos de recupe-

ração (PR) por coordenação entre os processos. Durante o processo de estabelecimento dos PRs, apenas as mensagens de controle do procedimento podem transitar pelo canal. Desta forma, cada conjunto de PRs locais (de cada processo) forma uma **linha consistente de recuperação**, possibilitando o descarte dos PRs anteriormente estabelecidos.

Mesmo tendo desaparecido a falha (temporária, por hipótese), um dos processos detecta a existência de erros resultantes e inicia o processo de recuperação. Este processo resume-se em atualizar os dados e o processamento a partir do último (e único) PR registrado.

O algoritmo utiliza um protocolo de duas fases (*two phase commit protocol*) para estabelecer um PR e retornar a um PR. Desta forma, uma tentativa de estabelecimento de um PR pode vir a falhar, sendo adiada; no caso do retorno, o algoritmo não adia o procedimento, ficando bloqueado até conseguir o retorno de todos os processos.

Deve-se notar que o mecanismo de recuperação envolve o descarte de uma porção de processamento, mais precisamente aquela compreendida entre o estabelecimento do último PR e a detecção da ocorrência da falha. Este fato implica queda de desempenho.

A escolha da periodicidade de estabelecimentos dos PRs inadequada aos parâmetros de falhas do sistema tem repercussões sobre o desempenho. Se o período entre o estabelecimento dos PRs for muito grande, em média muito processamento será desfeito a cada falha, levando a um baixo desempenho na ocorrência freqüente de falhas; caso o período seja muito pequeno, o sistema gastará muito tempo coordenando a tomada dos PRs, reduzindo significativamente o desempenho também, mesmo em operação normal (sem falhas).

3 Características do algoritmo assíncrono

No caso do algoritmo de Juang e Venkatesan [JUA91], não há coordenação entre os processos para a tomada de um PR. Cada processo, após o recebimento de uma mensagem, salva um PR em memória volátil (que pode ser perdido em caso de falha) e, de forma periódica, transfere estes dados para a memória estável (protegida contra falhas). O estabelecimentos destes PRs locais evita que a comunicação normal precise ser suspensa. Além disso, não existem as trocas de mensagens necessárias a coordenação.

Entretanto, quando o sistema detecta um erro, antes de iniciar o processo de retorno propriamente dito, deve ser encontrada uma linha consistente de recuperação entre os vários PRs locais armazenados nos processos: se o processo tiver falhado, deverão ser utilizados os dados da memória estável; em caso contrário, podem ser usados os dados da memória volátil.

O algoritmo assíncrono não está livre do "efeito dominó", mas é limitado ao último armazenamento em memória estável do processo que falhou. Entretanto, como as ações de salvamento não são difundidas aos demais processos, devem ser mantidos em memória os PRs obtidos desde que a aplicação foi iniciada.

Na literatura, é sustentado que, em operação normal, a queda de desempenho é relativamente menor no uso do algoritmo assíncrono se comparado ao síncrono; a situação inverte-se quando ocorrem falhas.

4 Suposições e princípios gerais

Algumas restrições foram necessárias para simplificar a análise de desempenho; outras foram necessárias para compatibilizar a operação dos dois algoritmos:

- não foi considerado o mecanismo recursivo proposto por Koo e Toueg no estabelecimento dos PRs e no retorno a um PR;
- foi considerado que as falhas ocorrem de forma periódica, sendo o período médio designado por TBF (*Time Between Faults*);
- todos os processos ativos serão considerados no procedimento de tolerância a falhas,

mesmo que não tenham trocado mensagens com o grupo de processos que falhou;

- não ocorrem falhas durante a execução dos algoritmos (tomada de PRs e retorno).

O *desempenho* calculado fornecerá o percentual do tempo utilizado para o processamento da aplicação sobre o tempo total gasto, o qual corresponde à soma dos tempos gastos pela aplicação e nas atividades de tolerância a falhas. Esta equação do desempenho é:

$$Desempenho = \frac{(TempoTotal - TempoTotalToleranciaFalhas)}{TempoTotal}$$

Como, por hipótese, as falhas ocorrem de forma periódica, o *desempenho* calculado a partir dos tempos totais será, em média, o mesmo obtido utilizando-se os tempos de um período entre falhas. Ou seja, o *desempenho* ou a esperança do *desempenho relativo* (DR) será:

$$E(DR) = E\left(\frac{(TBF - TempoToleranciaFalhas)}{TBF}\right) = 1 - \frac{E(TempoToleranciaFalhas)}{TBF}$$

O tempo gasto com as atividades de tolerância a falhas foi dividido em duas etapas: o estabelecimento dos PRs e o retorno a um PR, as quais dependem do algoritmo considerado.

5 Desempenho relativo do algoritmo síncrono

Considerando que T_{CP} é a periodicidade da tomada dos pontos de recuperação, T_{PR} é o tempo gasto para a tomada de um PR e T_{REC} o tempo gasto no processo de retorno, a equação que descreve o desempenho relativo do algoritmo síncrono é a seguinte:

$$E(DR) = \left(1 - \frac{E(T_{PR})}{T_{CP}}\right)\left(1 - \frac{E(T_{REC})}{TBF}\right)$$

Esta equação é formada por duas parcelas que contribuem para a queda no desempenho: uma devida ao tempo gasto no estabelecimento dos PRs e a outra devida ao processo de retorno a um PR. O tempo gasto no estabelecimento dos PRs é calculado pela equação:

$$E(T_{PR}) = T_{FIXO} + \left[P_{PR} \times E(T_{VAR}^{OK}) + (1 - P_{PR}) \times E(T_{VAR}^{NOK})\right],$$

onde T_{FIXO} é o tempo gasto no estabelecimento de qualquer PR, P_{PR} é a probabilidade de obter-se um PR, T_{VAR}^{OK} é o tempo adicional gasto quando é obtido um PR e T_{VAR}^{NOK} o tempo adicional gasto quando não é obtido o PR.

As parcelas componentes da esperança do tempo gasto no estabelecimento dos PRs podem ser calculadas pelas seguintes equações, onde N é o número de processos:

$T_{FIXO} = 3 \cdot t_m + (2 \cdot N - 2) \cdot t_{me}$, quando não há suporte para *broadcast* de mensagens;

$T_{FIXO} = 3 \cdot t_m + (3 \cdot N - 4) \cdot t_{me}$, quando há suporte para *broadcast* de mensagens;

$T_{VAR}^{OK} = T_{VAR}^{NOK} = T_{PRP}$, onde T_{PRP} é o tempo que um processo gasta para salvar os dados de um ponto de recuperação. Na realidade, T_{VAR}^{OK} e T_{VAR}^{NOK} não são iguais, mas a diferença entre eles é pequena; o tempo T_{VAR}^{OK} é maior pois inclui a ativação do PR na memória estável.

Para o tempo gasto no processo de retorno, tem-se a equação:

$$E(T_{REC}) = T_{CP} \times \left(\frac{2 - P_{PR}}{2 \times P_{PR}}\right) + T_{DET} + \frac{T_{FIXO}}{P_{RET}} + \left[\left(\frac{1 - P_{RET}}{P_{RET}}\right) \times T_{VAR}^{NOK} + T_{VAR}^{OK}\right],$$

onde P_{RET} é a probabilidade de que uma tentativa de retorno seja bem sucedida; T_{DET} é o tempo entre a manifestação da falha e sua detecção; T_{VAR}^{OK} é o tempo gasto em uma tentativa de retorno bem sucedida e T_{VAR}^{NOK} em tentativas sem sucesso (deve haver nova tentativa).

Os resultados obtidos são mostrados na forma de gráficos. Na figura 1, são apresentadas as curvas de desempenho em função (a) da periodicidade das falhas, *TBF* e (b) da periodicidade da tomada dos PRs, T_{CP}.

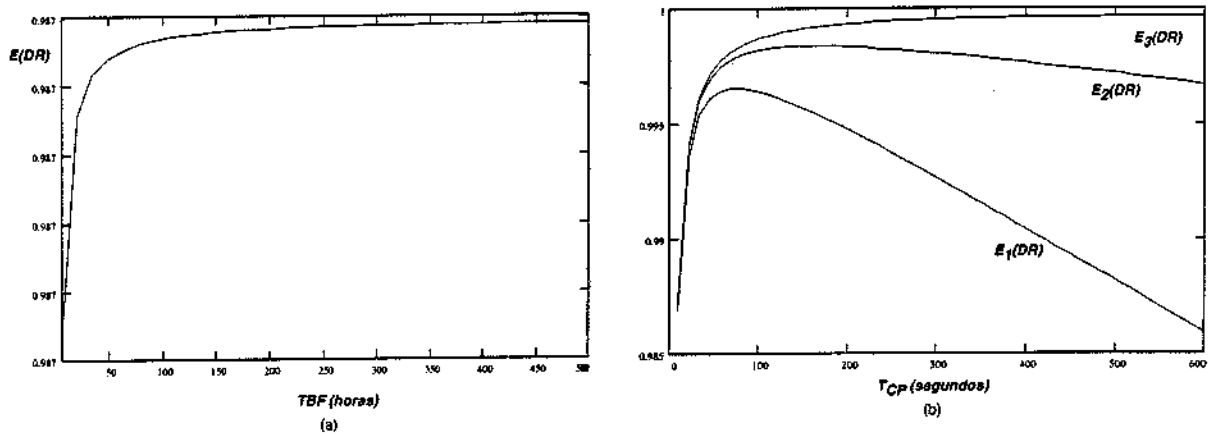


Figura 1 - Curvas de desempenho do algoritmo síncrono

O desempenho cresce, de forma assintótica, com o aumento de TBF. O valor da assíntota pode ser calculado pela aplicação de limite à equação de desempenho:

$$\lim_{TBF \to \infty} E(DR) = 1 - \frac{E(T_{PR})}{T_{CP}}$$

Na figura 1 foram apresentadas três curvas de desempenho. Cada curva foi traçada usando um valor diferente de *TBF*. A curva $E_1(DR)$ corresponde ao menor valor de *TBF* e a curva $E_3(DR)$ é aquela com o maior valor de *TBF*. Todas estas curvas apresentam um ponto de máximo, o qual pode ser calculado igualando a zero a derivada da equação de desempenho. Como resultado obtém-se:

$$T_{CP\,max} = \sqrt{\left[E(T_{PR}) \times (TBF - K2)\right] / K1}$$

$$\text{onde } K1 = \left(\frac{2 - P_{PR}}{2 \times P_{PR}}\right) \text{ e } K2 = T_{DET} + \frac{T_{FIXO}}{P_{RET}} + \left[\left(\frac{1 - P_{RET}}{P_{RET}}\right) \times T_{VAR}^{NOK} + T_{VAR}^{OK}\right].$$

6 Desempenho relativo do algoritmo assíncrono

Da mesma forma que a equação de desempenho do algoritmo síncrono, aparecem duas parcelas na equação de desempenho: uma devida ao processo de estabelecimento dos PRs e outra devida ao retorno a um PR, conforme equação que segue:

$$E(DR) = \left(1 - \frac{T_{PRV}}{1/\lambda} - \frac{T_{PRP}}{T_{CP}}\right)\left(1 - \frac{E(T_{REC})}{TBF}\right),$$

onde T_{PRV} corresponde ao tempo que o processo gasta para salvar o seu estado em memória volátil, T_{PRP} corresponde ao tempo gasto para salvar o estado do processo em memória estável e λ é a taxa média de recebimento de mensagens.

Na equação de *desempenho relativo*, pode-se notar que o tempo gasto nas atividades que degradam o desempenho aparecem divididas pela periodicidade que ocorrem: tempo gasto na escrita em memória volátil pelo tempo entre mensagens; tempo gasto na escrita em memória estável pelo tempo entre salvamentos de PRs; tempo gasto no retorno a um PR pela periodicidade das falhas. Os tempos gastos no salvamento do estado do processo em memória volátil ou memória estável são dependentes do hardware das memória envolvidas. Entretanto,

quanto maior for a relação entre o tempo de acesso à memória estável e o tempo de acesso à memória volátil, tanto melhor pode-se esperar que seja o desempenho.

O tempo gasto no processo de retorno pode ser calculado pela seguinte equação:

$$E(T_{REC}) = \frac{T_{CP}}{2} + T_{DET} + T_{BRC} + N \times (T_{TM} + T_{PM}),$$

onde T_{DET} é o tempo entre a manifestação da falha e a detecção deste fato; T_{BRC} é o tempo gasto para que o processo que falhou informe este fato aos outros processos do sistema; T_{TM} é o tempo gasto em uma rodada de busca de uma linha consistente de recuperação; T_{PM} é tempo gasto para processar as mensagens em uma destas rodadas. Como pode ser observado na equação, são necessárias N rodadas para que seja obtida a linha de recuperação. A equação do cálculo de T_{TM} é a seguinte:

$$T_{TM} = N \times [t_m + (N-2) \times t_{me}]$$

A representação gráfica das equações obtidas para o desempenho relativo pode ser vista na figura 2, mostrando a relação entre o desempenho relativo e (a) a periodicidade das falhas, *TBF*, e (b) a periodicidade da tomada dos PRs, T_{CP}. As curvas em 2(b) foram traçadas para valores diferentes de *TBF*: $E_1(DR)$ para o menor valor e $E_3(DR)$ para o maior valor.

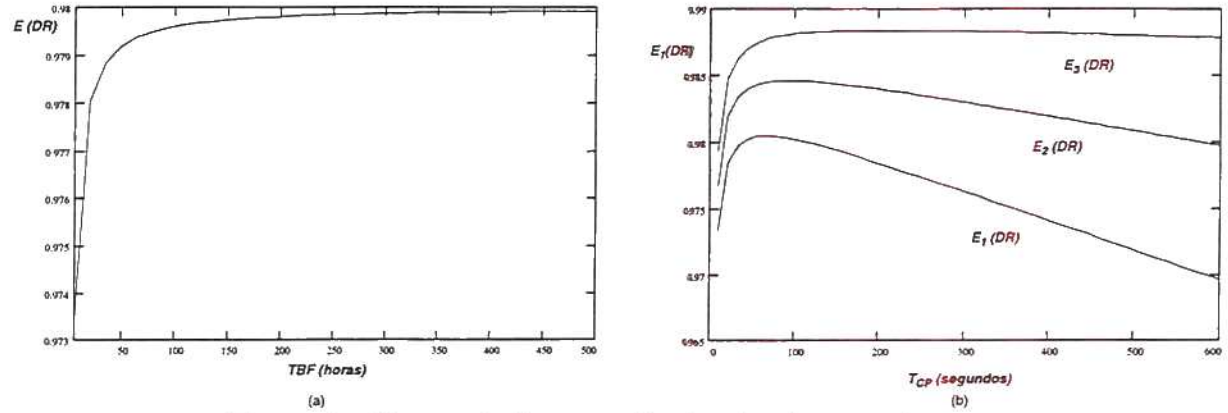


Figura 2 - Curvas de desempenho do algoritmo assíncrono

A forma geral das curvas de desempenho é a mesma que as do algoritmo síncrono. O desempenho cresce de forma assintótica, quando *TBF* aumenta, para um valor que pode ser calculado pelo limite:

$$\lim_{TBF \to \infty} E(DR) = 1 - \frac{T_{PRV}}{1/\lambda} - \frac{T_{PRP}}{T_{CP}}.$$

Através deste resultado pode-se verificar que, mesmo com uma baixa taxa de falhas (*TBF* grande), o desempenho não poderá passar de um valor máximo. Este valor depende do tempo gasto para salvar um PR: volátil e estável; e da periodicidade destes salvamentos: taxa de recebimento de mensagens e taxa de transferência para a memória estável.

A curva de desempenho em função de T_{CP} apresenta um ponto de máximo. Este pode ser calculado igualando-se a zero a derivada da função de desempenho:

$$T_{CP\,max} = \sqrt{[T_{PRP} \times (TBF - K2)] / \left[K1 \times \left(1 - \frac{T_{PRV}}{1/\lambda} \right) \right]}$$

onde $K1 = \frac{1}{2}$ e $K2 = T_{DET} + T_{BRC} + N \times (T_{TM} + T_{PM})$.

7 Análise e Conclusões

A comparação das equações obtidas levaram às conclusões a seguir relatadas.

Para ambos algoritmos, a medida que a taxa de falhas diminui, o fator básico determinante do desempenho é o período da tomada dos PRs (TCP). Entretanto, o desempenho do algoritmo assíncrono piora com o aumento do número de mensagens recebidas, λ.

O aumento do número de processos é mais prejudicial ao desempenho no algoritmo assíncrono; as equações apresentam um fator de redução de desempenho N^3 para o algoritmo assíncrono e N para o síncrono.

Como os processos comunicam-se só através de mensagens, os tempos destas foram considerados. Seus efeitos aparecem em todas as etapas do algoritmo síncrono e na recuperação do algoritmo assíncrono, onde aparece multiplicada pelo fator N^3.

A existência de suporte para *broadcast* (ou uma arquitetura que o privilegie) é importante no estabelecimento dos PRs do algoritmo síncrono e na recuperação do assíncrono.

Além dos exposto, a taxa de recebimento de mensagens só influencia o desempenho do algoritmo assíncrono e pode ser decisivo na escolha do algoritmo. Esta comparação pode ser observada na figura 3, onde foram traçadas as curvas de desempenho para os algoritmos síncrono ($E_S(DR)$) e assíncrono com λ=1 ($E_{A1}(DR)$) e λ=5 ($E_{A2}(DR)$).

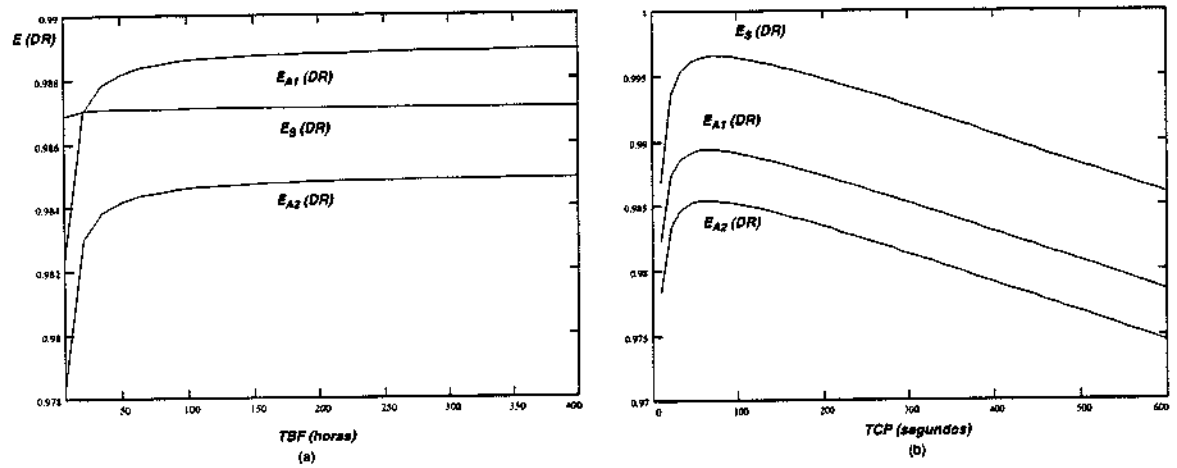


Figura 3 - Curvas de desempenho comparativas

Apesar dos resultados coincidirem com as previsões apresentadas na literatura, é importante que sejam validados através da implementação ou da simulação dos algoritmos, o que ainda não foi feito.

O estudo aprofundado dos algoritmos e das hipóteses adotadas para a modelagem matemática, bem como a dedução de todas as equações, estão disponíveis através da monografia [CEC98] preparada como parte das atividades preliminares ao doutorado no CPGCC.

Referências

[CEC98] Cechin, S. L. Avaliação teórica do desempenho de algoritmos de recuperação por retorno do tipo síncrono e assíncrono. CPGCC da UFRGS. 1998.

[JUA91] Juang, T.; Venkatesan, S. Crash Recovery with Little Overhead. Int'l. Conf. on Distributed Computing Systems. Proceedings. May 1991. Pp.454-461.

[JAL94] Jalote, P. *Fault Tolerance in Distributed Systems*. New Jersey: Prentice-Hall, 1994.

[KOO87] KOO, R; TOUEG, S. Checkpointing and Rollback-Recovery for Distributed Systems. *IEEE Trans. on Software Engineering*, v.SE-13(1):23-31, Jan. 1987.

Abordagens de Comunicação em Sistemas Distribuídos Tempo Real

Patrícia Pitthan de Araújo Barcelos[1] Taisy Silva Weber[2]

Universidade Federal do Rio Grande do Sul
Instituto de Informática
Caixa Postal 15064, CEP 91501-970
Porto Alegre – RS, Brasil
{pitthan, taisy@inf.ufrgs.br}

Resumo

Confiabilidade em sistemas distribuídos está deixando de ser uma característica desejável para se tornar uma necessidade. Este fato é ainda mais marcante em se tratando de sistemas distribuídos tempo real, cujo cumprimento das exigências temporais impostas está diretamente relacionado às conseqüências da ocorrência de uma falha. Este artigo descreve algumas das características, exigências e garantias de sistemas distribuídos tempo real sob o ponto de vista de comunicação, enfatizando sua aplicação em um sistema de controle distribuído.

Abstract

Distributed systems reliability are becoming necessary instead of a desirable feature. This fact is more important if we're talking about real time distributed systems, whose execution of temporal requirementes imposed is closely related to the results of a fault. This article describes some real time distributed systems features, requirements and guarantees under communication point of view, emphasizing its application in a distributed control system.

1. Introdução

Quando se deseja controlar o comportamento de um sistema deve-se agir sobre o mesmo em momentos determinados do tempo. Desta forma é possível realizar uma análise dos efeitos produzidos pela ação tomada. Tal situação ilustra o funcionamento de um sistema tempo real. Apesar de redundante, a definição informal de sistema tempo real pressupõe um sistema que muda seu estado em função do tempo, o tempo real.

Atualmente os sistemas tempo real apresentam-se cada vez mais de forma distribuída. Tais sistemas são compostos por um conjunto de nodos interconectados por um sistema de comunicação tempo real. O acesso a esses sistemas de comunicação se dá por meio de protocolos tempo real, os quais caracterizam-se por apresentar um tempo de execução máximo reduzido a um período conhecido.

---

[1] Mestre e Doutoranda em Ciência da Computação (UFRGS), Professora do Instituto de Informática da PUCRS. Áreas de interesse: Sistemas Distribuídos, Redes de Computadores, Tolerância a Falhas.

[2] Doutora em Ciência da Computação (Karlsuhe, 1986), Professora do Departamento de Informática da UFRGS. Áreas de Interesse: Sistemas Distribuídos, Tolerância a Falhas, Arquitetura de Computadores.

Este artigo apresenta aspectos de comunicação em sistemas distribuídos tempo real. São explorados os conceitos básicos de sistemas tempo real, que envolvem sua definição, classificação e paradigmas de implementação. A forma com que os sistemas tempo real endereçam as características de sincronismo na presença ou não de relógios, bem como a maneira pela qual manipulam mensagens frente às exigências temporais impostas também são alvo de análise. Com base neste estudo, é proposta uma aplicação para a qual os aspectos acima mencionados podem ser associados.

O objetivo deste trabalho é apresentar os resultados de uma pesquisa em comunicação em sistemas tempo real enfatizando uma aplicação de controle para a qual é necessária a manipulação de restrições temporais.

2. Sistemas Distribuídos Tempo Real

Kopetz [KOP 93] define um sistema tempo real como um sistema que muda de estado em função do tempo (real). Esta é uma definição genérica e engloba todo e qualquer sistema tempo real, não somente aqueles controlados por sistemas computacionais.

Para Santos [SAN 91], um sistema é caracterizado como tempo real porque as aplicações que ele controla devem ocorrer em um determinado tempo, de forma independente do sistema controlador. Assim, o relógio que controla as ações do sistema tempo real não é o relógio interno do computador, mas sim o relógio que controla a dinâmica dos estados do processo no ambiente em que o sistema está inserido.

Uma das maiores dificuldades encontradas no desenvolvimento de sistemas tempo real é que eles, normalmente, são caros, pois exigem um profundo conhecimento sobre a aplicação onde atuam. Além disso, são difíceis de testar, uma vez que devem ser verificados diretamente ou por meio de simulação. Em ambos os casos, pequenas alterações no sistema acarretam em uma nova rodada de testes. Sistemas tempo real diferem de sistemas convencionais por apresentarem restrições temporais e tratarem, na maioria das aplicações, com situações críticas. Em tais aplicações, a ocorrência de qualquer tipo de falha, inclusive falha de temporização, pode causar conseqüências catastróficas. Portanto, ao contrário dos sistemas onde há uma separação entre correção e desempenho, em sistemas tempo real, correção e desempenho estão fortemente relacionados [STA 88].

Uma classificação de sistemas tempo real amplamente aceita na literatura é proposta por Kopetz e Veríssimo [KOP 93]. Segundo eles, sistemas tempo real são classificados como sistemas tempo real brandos e sistemas tempo real críticos. Tal classificação leva em consideração o atendimento aos requisitos temporais e a conseqüência das falhas apresentadas pelos sistemas.

Um sistema é caracterizado como tempo real brando se as aplicações que ele controla são capazes de tolerar pequenos atrasos, os quais não implicam em conseqüências desastrosas. Este sistema pode apresentar exigências de alto grau de disponibilidade de utilização e de integridade física no que se refere aos dados. Quando as conseqüências de uma falha conduzem a situações de risco, trata-se de um sistema tempo real crítico. Tais sistemas podem se apresentar livres de falhas ou com falhas mascaradas. Um sistema tempo real crítico é considerado livre de falhas se um ou mais estados, os quais serão acessados em caso de falha, podem ser identificados. Entretanto, se a aplicação não permite a identificação de estados seguros, o sistema deve fornecer um nível mínimo de serviço mesmo no caso de falha,

evitando, com isso, a ocorrência de uma catástrofe. Este sistema é caracterizado por apresentar falha mascarada.

Quanto à abordagem seguida durante a implementação, os sistemas tempo-real podem ser classificados ainda como sistemas tempo real com respostas garantidas (*guaranteed-response*) ou sistemas tempo real de melhor esforço (*best-effort*). Se, a partir de cenários de carga e falhas, cria-se um projeto que torna possível um debate sobre sua adequação sem usar argumentos probabilísticos, diz-se que o sistema possui resposta garantida. Os sistemas tempo real de melhor esforço são projetados especialmente para atuarem em situações onde é fundamental que o controle seja completo, irrestrito e confiável, visando garantir a disponibilidade do sistema, mesmo quando é necessária uma determinada adaptabilidade às mudanças do ambiente, como no caso de controle de tráfego aéreo.

Em sistemas tempo real há ainda que se considerar os paradigmas sob os quais os mesmos são implementados. Quanto à forma de reação às modificações externas, existem dois paradigmas distintos utilizados no projeto de sistemas tempo real: sistemas disparados por evento (*event-triggered*) e sistemas disparados por tempo (*time-triggered*) [KOP 94].

No caso de sistemas disparados por evento, todas as atividades são iniciadas como conseqüência de eventos externos, ou seja, alguma mudança significativa do estado do sistema. A reação a eventos externos deve ocorrer de forma direta e imediata, de forma a não comprometer a característica tempo real do sistema. Tais sistemas são caracteristicamente não previsíveis, uma vez que não é possível determinar antecipadamente quando serão realizadas as ações. Entretanto, alguma previsibilidade pode ser alcançada caso seja possível a limitação da taxa máxima de eventos em função da física do sistema controlado.

Em sistemas disparados por tempo todas as atividades são dirigidas pela progressão do tempo global. Desta forma, todas as tarefas ou ações de comunicação do sistema ocorrem em instantes de tempo pré-determinados. Este paradigma é menos flexível que o anterior, porém, devido a característica de previsibilidade, é mais fácil de ser analisado e testado. Sistemas disparados por tempo exigem sincronismo entre os nodos do sistema distribuído no qual estão inseridos.

3. Comunicação Síncrona x Assíncrona

Protocolos de comunicação tempo real seguem a mesma terminologia adotada pelos sistemas convencionais, onde são encontrados protocolos síncronos e protocolos assíncronos.

Protocolos síncronos são necessários para aplicações tempo real críticas, as quais devem obedecer limites de tempo de resposta mesmo quando falhas de componentes ocorrem. Tais limites são alcançados pela suposição de que atrasos de mensagens entre processadores corretos são limitados e há suficientes caminhos de comunicação redundantes entre os processadores tal que, se todos os caminhos são usados em paralelo para uma difusão, a probabilidade que todos falhem durante a difusão é insignificante [KOP 92].

A suposição de atrasos exige que processadores que implementam um protocolo síncrono sejam controlados por sistemas operacionais tempo real capazes de garantir limites nos atrasos de escalonamento de tarefas. Quando mensagens não são limitadas, isto é, falhas de comunicação podem ocasionar particionamentos, a terminação da difusão em um tempo limitado não pode ser garantida [KOP 92].

Protocolos assíncronos sacrificam terminação para oferecer tolerância a falhas de comunicação, incluindo falhas que conduzem ao particionamento. Por não garantirem limitação no tempo ao difundir uma informação na presença de falhas, tais protocolos não podem ser usados em aplicações críticas, as quais devem garantir que limites temporais sejam sempre encontrados, mesmo na ocorrência de falhas de um componente[KOP 94].

4. Serviços de Comunicação

As semânticas dos serviços de comunicação tempo real podem ser caracterizadas pelas combinações das propriedades de concordância, ordenação e sincronismo. Há essencialmente duas maneiras de classificar protocolos de comunicação de grupo, no que se refere às propriedades do domínio do tempo: protocolos disparados por relógio (*clock-driven*) e protocolos sem relógio (*clockless*).

Protocolos disparados por relógio contam com a existência de um tempo global baseado em relógios ([BAB 85], [CRI 90], [CRI 85]), enquanto os protocolos sem relógio utilizam-se de referências de tempo relativas, os *timers* ([BIR 87], [KAA 89], [VER 90]).

Protocolos disparados por relógio e protocolos sem relógio são classificados em síncronos e assíncronos, respectivamente, embora a primeira terminologia seja preferida, uma vez que existem protocolos síncronos sem relógio. Os protocolos síncronos sem relógio apresentam como característica a presença de limitação nos atrasos das mensagens e capacidade de ordenação temporal das mesmas [VER 90].

5. Mensagens em Sistemas de Comunicação Tempo Real

As aplicações em sistemas de comunicação tempo real podem ser bastante distintas quanto ao tipo de restrição temporal, escala de tempo envolvida, distribuição de controle, âmbito de atuação, etc. Estes fatores determinam diferentes requisitos de comunicação e, portanto, as formas de atendimento a estes requisitos podem exigir várias funcionalidades para o seu suporte de comunicação tais como escalonamento de mensagens, reserva de recursos, controle de tráfego da rede, nem sempre necessárias em um sistema de comunicação convencional [ARV 93].

As mensagens em sistemas de comunicação tempo real são caracterizadas por duas restrições temporais, ocorrência no sistema e exigências quanto a sua integridade física. As mensagens com restrições de tempo podem ser classificadas segundo suas exigências de garantia [ARV 93]: mensagens que exigem garantia (*guarantee seeking*) e mensagens com restrições de tempo, mas que não requerem uma garantia quanto ao seu cumprimento (*best-effort*).

A primeira classe de mensagens, mensagens que exigem garantia (*guarantee seeking*), se refere a mensagens críticas, ou seja, essenciais para a operação correta do sistema tempo real. O sistema deve incluir uma garantia de que a atividade referida pela mensagem teve execução aceita e suas restrições de tempo foram obedecidas com certeza. A segunda classe de mensagens, mensagens com restrições de tempo que não exigem garantia quanto ao seu cumprimento (*best-effort*), corresponde às aplicações nas quais as restrições de tempo associadas não implicam em alto custo com relação aos benefícios da operação normal. Neste caso, o sistema de comunicação deve tentar satisfazer as restrições minimizando o número

demensagens com restrições de tempo violadas. Um percentual de perdas ou atrasos é admitido para este tipo de mensagem.

6. Aplicação da Comunicação em Sistemas Tempo Real

Os aspectos enfocados neste artigo denotam as características usualmente necessárias em sistemas distribuídos tempo real. Entretanto, a apresentação de tais características implica na determinação de uma aplicação que envolva as mesmas. Uma das aplicações para a qual são necessárias restrições temporais corresponde a um sistema de controle distribuído de tráfego ferroviário. Tal sistema pode ser classificado como sistema tempo real de melhor esforço (*best-effort*).

A aplicação de controle de tráfego ferroviário baseia-se em um experimento desenvolvido em Newcastle por Cecília Rubira [RUB 94], cuja implementação foi modelada com orientação a objetos, entretanto não considera restrições temporais e leva em conta apenas falhas de ambiente. Como o nosso objetivo não está concentrado em modelagem, a idéia é utilizar esta sugestão de aplicação para implementar comunicação de grupo com restrições temporais.

O sistema de controle distribuído de tráfego ferroviário proposto supõe uma malha ferroviária composta pelos seguintes elementos:

- vias, as quais por definição são todas unidirecionais, formadas por trilhos, sendo que seções de trilhos podem ser compartilhadas por mais de uma via;
- cruzamentos controláveis, os quais possuem chaves que recebem o sinal de rota;
- trens inteligentes, não somente com capacidade de enviar e receber informações do sistema, mas também com capacidade de ser controlados pelo sistema ou, em caso de colapso total do sistema de controle, procurar um estado seguro;
- sistema distribuído de contole de vias;
- sensores de fluxo nas vias, os quais informam a presença ou não de trem no trilho.

Cada seção de trilhos é controlada por um grupo de nodos (processadores do sistema). Além disso, os grupos devem comunicar-se para avisar a passagens de trens pela fronteira entre as seções. Neste sentido são necessários protocolos de comunicação de grupo, as quais suportem restrições temporais, como os apresentados itens 3 e 4.

Os objetivos do sistema de controle são evitar colisões e descarrilamento de trens e fornecer informações aos trens sobre rotas alternativas. O modelo de falhas para este sistema pode ser composto por falhas de ambiente e falhas do sistema. Situações que ilustram falhas de ambiente são vias bloqueadas, chaves e sensores com defeito, e falta de comunicação com trens. Falhas do sistema podem ocorrer quando não há comunicação entre sistema e chaves ou sensores, servidores ou controladores não respondem a ativações do sistema e falha na comunicação entre os nodos.

O tráfego ferroviário pode ser considerado um sistema de controle relativamente fácil de ser representado, uma vez que as vias férreas apresentam grau de liberdade limitado e a taxa de defeitos de ambiente é baixa. É interessante observar que não pode ser assumido o modelo de falha simples (ou seja apenas uma falha de cada vez) devido a grande quantidade de elementos no sistema. Para cada componente que pode apresentar defeito, deve ser analisado qual o comportamento sob falha assumido. Além disso, deve ser analisado também o impacto de falhas de temporização no comportamento do sistema.

7. Conclusão

Uma característica quase sempre presente em sistemas tempo real é a previsibilidade, que indica a capacidade de se prever uma violação de garantia ou uma incorreção temporal. Esta característica torna possível um tratamento de exceções que minimize as conseqüências de uma falha, desde que, obviamente, esta previsão aconteça em tempo hábil para iniciá-lo.

Em um ambiente distribuído, como no caso da aplicação de controle ferroviário, a obtenção de uma garantia de correção temporal e de previsibilidade exige diversos cuidados na especificação e configuração do suporte de comunicação. A configuração do suporte requer uma escolha criteriosa dos parâmetros do sistema de comunicação, que envolve a determinação de protocolos e serviços de comunicação. Além disso, a configuração do suporte deve considerar a carga do sistema de comunicação e a arquitetura de comunicação adotada para o sistema.

A modelagem de aplicação exposta servirá de base para a construção do ambiente de teste para o sistema de controle, o qual possivelmente será implementado a partir de uma rede convencional, composta por PC's e sistema operacional QNX, caracteristicamente tempo real.

Comunicação em sistemas distribuídos tempo real tem sido alvo de diversas pesquisas [ARV 93] [FET 97] [FON 93] [KOP 93] [KOP 94] [KOY 88]. Entretanto, por se tratar de um assunto cujo interesse vem aumentando gradativamente, há uma série de tópicos relacionados ainda não explorados.

8. Referências Bibliográficas

[ARV 93] ARVIND, K. et. al. *A Local Area Network Architecture for Communication in Distributed Real Time Systems. In*: Advances in Real Time Systems, IEEE, Los Alamitos, California, 1993.

[BAB 85] BABAOGLU, O.; DRUMMOND, R. *Streets of Byzantium: Network Architectures for Fast Reliable Broadcast*. IEEE TOSE, v.11, n.6, 1985.

[BIR 87] BIRMAN, K.; JOSEPH, T. *Reliable Communication in the Presence of Failures*. ACM TOCS, NY, v. 5, n. 1, Feb. 1987.

[CRI 85] CRISTIAN, F. et. al. *Atomic Broadcast: From Simple Message Diffusion to Byzantine Agreement. In*: XV FTCS (Fault Tolerant Computer Systems), Ann Arbor, USA, 1985.

[CRI 90] CRISTIAN, F. *Synchronous Atomic Broadcast for Redundant Broadcast Channels*. Journal of Real Time Systems, NY, v. 2, n. 3, Sep. 1990.

[FET 97] FETZER, C.; CRISTIAN, F. *Real Time Systems*. Distributed Computing, Berlin, 1997.

[FON 93] FONSECA, K. O.; FARINES, J-M. *Uma Análise das Diversas Propostas de Atendimento dos Requisitos de Comunicação para Sistemas Tempo Real em Sistemas de Manufatura*. XI SBRC,1993.

[KAA 89] KAASHOEK, M. F. et. al. *An efficient reliable broadcast protocol*. Operating System Review, v. 2, 1989.

[KOP 92] KOPETZ, H. *Sparse Time versus Dense Time in Distributed Real Time Systems. In*: International Conference on Distributed Computing Systems, Yokohama, Japan, 1992.

[KOP 93] KOPETZ, H.; VERÍSSIMO, P. *Real Time and Dependability Concepts. In*: Mullender, S. J. (Ed.) Distributed Systems. NY, 1993.

[KOP 94] KOPETZ, H. *Protocol for Real Time Systems*. Computer, NY: IEEE, Jan. 1994.

[KOY 88] KOYMANS, R. et. al. *Paradigms for Real Time Systems. In*: Symposium on Formal Technicques in Real Time and Fault-Tolerant Systems, Warwick, Gran Bretanha, 1988.

[SAN 91] SANTOS, J. *Redes Locales en Tiempo Real*. Nova Friburgo: EBAI, 1991.

[RUB 94] RUBIRA, C. M. F. *Structuring Fault-Tolerant Object-Oriented Systems Using Inheritance and Delegation*. PhD Thesis, University of Newcastle upon Tyne, 1994.

[STA 88] STANKOVIC, J. A.; RAMAMRITHAM, K. *Hard Real Time Systems*. NY: IEEE, 1988.

[VER 90] VERÍSSIMO, P.; MARQUES, J. *Reliable broadcast for fault-tolerance on local computer networks. In*: IX Symposium on Reliable Distributed Systems, Huntsville: IEEE, 1990.

Um Serviço Configurável de Sincronização de Relógios para o Sistema Operacional QNX

Alessandro Dario Agnoletto
Taisy Silva Weber

Universidade Federal do Rio Grande do Sul
Instituto de Informática
Porto Alegre – Brasil
{agnolett,taisy}@inf.ufrgs.br

Resumo

O presente trabalho propõe um serviço de sincronização de relógios para o sistema operacional QNX [QNX93]. O serviço proposto deve poder ser ajustado para o tipo de aplicação de tempo real e ter acesso a uma fonte de tempo UTC através de um Receptor de Tempo GPS [DAN97]. O projeto inclui a implementação de um protocolo probabilístico [CRI89] e um protocolo determinístico [LAM85]. Os dois protocolos visam à sincronização externa, mas podem sofrer uma degradação para a sincronização interna (somente) quando não for possível o acesso ao tempo fornecido pela constelação GPS devido à falha desta ou do receptor. Pretende-se dispor este serviço para um sistema distribuído provido de sistema operacional QNX e rede Ethernet [TAN96]. Um protocolo de sincronização de relógios determinístico deve ser suportado por uma arquitetura de rede com atraso limitado no tempo de viagem de uma mensagem. Portanto, um anel lógico com passagem de *token* deve ser construído sobre a arquitetura nativa a fim de torná-la determinística. Com a construção do anel, devem ser levados em conta os efeitos colaterais ao funcionamento do sistema operacional.

Abstract

This paper considers a clock sinchronization service for QNX operating system [QNX93]. The proposed service must be configurable according to the type of the real-time application and must have access to a UTC time source through a GPS Time Receiver [DAN97]. The design includes the implementation of a deterministic protocol [LAM85] and a probabilistic one [CRI89]. The two protocols aim external sinchronization, but are able to degradate to internal sinchronization (only) when the time provided by the GPS constelation is not available due to fault of this or from the receiver. It is intended to offer this service for a distributed system provided with QNX operating system and Ethernet [TAN96]. A clock synchronization deterministic protocol must be supported by a network architecture with limited message delay. Therefore, a logical ring with token passing must be placed over the native architecture to become deterministic. With the construction of the ring, the collateral effects to the functioning of the operating system must be studied.

1. Introdução

O surgimento dos sistemas distribuídos acabou com a existência de uma fonte comum de tempo para ordenação e cronometragem a todos os processos de um sistema, visto que estes podem se encontrar em máquinas diferentes. A solução para o problema consiste em usar o relógio físico local a cada máquina para manter, em cada uma, um relógio virtual. Os relógios virtuais, por sua vez, são sincronizados entre si por um *algoritmo de sincronização de relógios*.

Tal sincronização viabiliza a implementação de aplicações distribuídas que realizam atividades dependentes do tempo, tolerantes a falhas ou não. As aplicações tolerantes a falhas utilizam o serviço de sincronização como bloco básico para a construção de protocolos confiáveis [JAL94]. Para sistemas de tempo real, então, a sincronização de relógios pode ser vital para o sincronismo e ordenação de atividades entre processos. Em algumas destas aplicações, a diferença entre o tempo marcado por dois relógios (sincronização interna [JAL94][AGN97]) deve ser sempre mantida dentro um limite máximo pré-determinado (com tal limite dependendo da aplicação em questão). Para outras aplicações, a sincronização interna deve apenas ser procurada (não obrigatoriamente mantida sempre dentro do limite de diferença). Além disto, pode ser desejada uma sincronização relativa a uma fonte de tempo oficial (sincronização externa [JAL94][AGN97]).

Normalmente, as propostas de protocolos de sincronização de relógios consideram apenas um tipo de arquitetura de rede e, portanto, se classificam como protocolos de sincronização determinística (precisão e/ou acuidade garantida(s)) ou probabilística (precisão e/ou acuidade não garantida(s)) [JAL94][AGN97]. Em nenhum caso ainda conhecido consideram um serviço que possa ser configurado de acordo com o tipo de arquitetura que será utilizado a partir do início das atividades do sistema.

O aproveitamento das características previamente existentes em sistemas operacionais de tempo real também nunca foi encontrado na literatura. Tal espécie de sistema operacional pode oferecer vantagens para a criação de um serviço de sincronização de relógios. Pela falta de tal iniciativa, não existem resultados sobre a exploração de tais.

O surgimento do Global Positioning System (GPS) [DAN97] deu novo impulso à área de sincronização de relógios. Isso se deve ao seu uso como fonte de disseminação de tempo oficial acessível praticamente em qualquer ponto da superfície terrestre que possa "avistar" os satélites do sistema. Tal fato é importante para quando a sincronização relativa a uma fonte de tempo UTC (Universal Time Coordinated) [DAN97] é necessária. O barateamento dos modelos mais simples de receptores GPS (inclusive alguns dedicados à recepção de tempo) permite atualmente que um serviço de sincronização de relógios tenha acesso a uma fonte de tempo oficial.

Por fim, existe o crescimento da necessidade de sistemas que supervisionem ambientes onde as tarefas precisam atender restrições temporais. Isso causa uma utilização maior de sistemas de tempo real, os quais, por sua vez, muitas vezes se baseiam em sistemas operacionais com características adequadas a tais restrições temporais. Neste quadro, o sistema operacional QNX se insere. Contudo, alguns serviços faltam a este sistema operacional e, entre eles, se encontra a falta de um serviço de sincronização de relógios.

2. O Sistema de Posicionamento Global

O Sistema de Posicionamento Global (GPS) [DAN97] é um sistema de navegação baseado em satélites orbitais que fornece, 24 horas por dia, posição tridimensional e tempo

precisos. GPS permanece sendo operado pela Força Aérea dos Estados Unidos sob orientação do Departamento de Defesa.

Os 10 primeiros satélites que compõem o sistema GPS foram lançados em 1989 e foram denominados Bloco I. Já em 1990, 43 laboratórios estavam utilizando GPS para sincronizar seus relógios atômicos. Em 1994, 24 satélites constituintes dos blocos II e IIA foram lançados. Estes 24 satélites formam a capacidade total do sistema GPS atualmente.

O Plano de Radionavegação Federal define dois tipos de serviços fornecidos pelo sistema:

• Standard Positioning Service: cujas acuidades de posicionamento e temporização são de 95%; isto significa que a acuidade do relógio recebido se encontra dentro de 340 nanosegundos; é de livre acesso a qualquer usuário;

• Precise Positioning Service: é limitado a usuários especificadamente autorizados pelos Estados Unidos; a acuidade do relógio se encontra dentro dos 200 nanosegundos.

O sistema GPS constitui-se de três segmentos:

• Segmento de Controle

Consiste das seguintes Estações Monitoras (figura 1): Ascension Island - Colônia Britânica no Oceano Atlântico Sul, Colorado Springs, Colorado [Master Control], Diego Garcia - base militar dos Estados Unidos no Oceano Índico, Hawaii e Atol de Kwajalein - República das Ilhas Marshall.

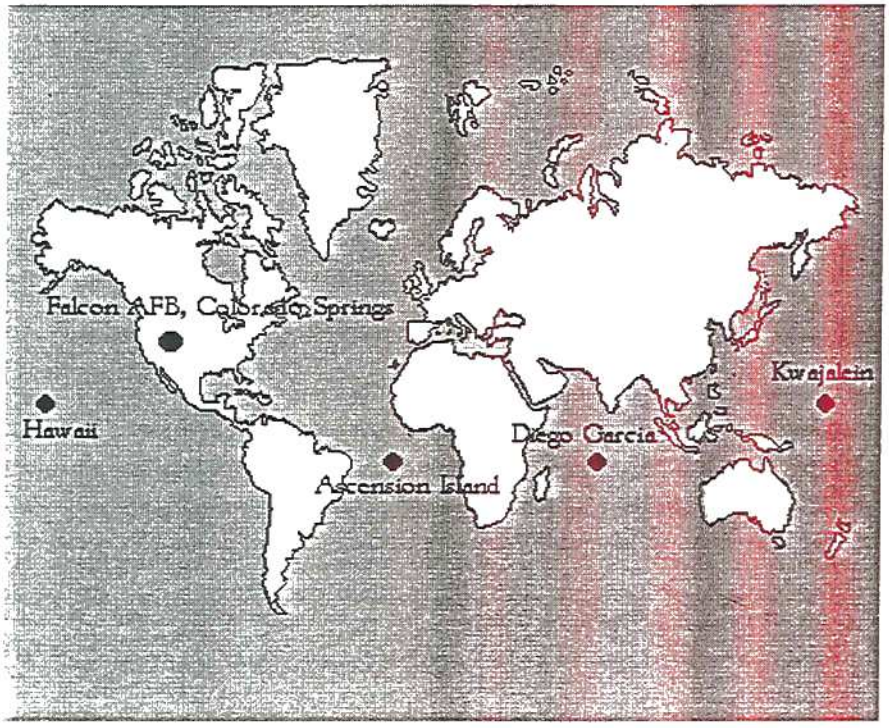


Figura 1: Estações Monitoras da constelação GPS

Estas estações rastream os satélites GPS que passam sobre elas duas vezes ao dia. Tais estações repassam a posição do satélite e a sua medida de tempo para o Controle Mestre existente na base da Força Aérea, no Colorado. Neste local os dados de todos os satélites são computados, ajustados e devolvidos para que as estações realimentem os satélites.

• Segmento de Espaço

É a constelação de 24 satélites que orbitam a 20.200 Km de altitude, dos quais 3 são reservas ativos. Estes satélites estão dispostos em seis planos orbitais, inclinados num angulo de 55 graus de diferença uns dos outros, ocasionando uma cobertura tal que em qualquer parte do mundo pode-se visualizar no mínimo 5 satélites.

Cada satélite transmite sinais de navegação em duas freqüências de microondas, as quais são moduladas em fase para a conter a Mensagem de Navegação, a qual consiste de dados orbitais, deslocamento do relógio do satélite e outros dados.

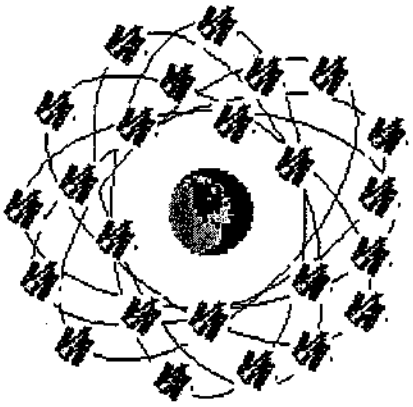


Figura 2: 24 satélites formadores da constelação GPS em seus 6 planos orbitais

• Segmento do Usuário

Consiste dos receptores do sinal GPS, militares e civis.

2.1. Controle do Tempo GPS

O controle do tempo é orientado pelo Controle Mestre para estar dentro de 1 microsegundo dentro do tempo UTC. O tempo GPS é derivado a partir do Relógio Composto GPS (Composite Clock - CC), o qual consiste de um relógio atômico em cada Estação Monitora e todos os padrões de freqüência dos satélites. Cada um dos satélites do Bloco II contém dois relógios de césio e dois relógios de rubídio.

O Observatório Naval dos Estados Unidos (USNO) monitora os sinais dos satélites diariamente, recolhendo os dados de temporização em 130 blocos de seis segundos. Estes 780 segundos formam uma Mensagem de Navegação completa (12,5 minutos), contendo a correção do relógio do satélite em relação ao UTC, ou seja, a diferença entre os dois tempos.

Os dados de temporização do GPS são comparados ao Master Clock do USNO, o qual é um conjunto de 60 relógios de césio e sete relógios de hidrogênio. Após comparados, os conjuntos de dados dos satélites são usados para ajustar os relógios do CC em uma taxa de 10^{-18} segundos por segundo ao quadrado.

O sinal GPS de um satélite é transmitido sob controle dos relógios atômicos daquele satélite. O satélite é monitorado e a diferença entre o tempo do Master Clock e o tempo do satélite é carregado para o veículo espacial. A seguir, o satélite envia para o receptor do usuário o fator de correção do seu relógio em relação ao tempo real.

Os sinais GPS estão sujeitos a diversas fontes de erros. Dentre estas, a mais destacada é o atraso ionosférico, o qual é o atraso da portadora dos sinais quando eles atravessam a camada de íons e elétrons livres conhecida como ionosfera. Variando em densidade e espessura (50 a 500 quilômetros) devido à pressão solar e efeitos geomagnéticos, a ionosfera pode causar atrasos de até 300 nanosegundos. Também a posição do satélite pode modificar o atraso do sinal, visto que quanto mais próximo o satélite estiver da linha do horizonte, mais

ionosfera o sinal terá de atravessar. Os satélites recebem informações sobre a ionosfera, a qual é transmitida para os receptores dos usuários.

Em oposição às antenas receptoras de sinal terrestre, uma antena GPS requer sempre contato "visual" com o satélite, ou seja, não podem existir obstáculos entre o mesmo e a antena [LIC97]. Ao receberem o sinal GPS, tais antenas estimam o atraso da transmissão. Tal atraso é corrigido ao adicionar-se o atraso da propagação pela antena o atraso interno ao receptor, as estimativas dos atrasos ionosférico e troposférico e outros efeitos relativísticos [DAN97].

2.2. Receptores de tempo GPS

Receptores de tempo GPS são projetados para o controle ou medição de tempo e intervalos de tempo. Esses receptores geralmente processam os sinais GPS usando técnicas especiais que assumem uma posição fixa e conhecida.

A acuidade de medidas de tempo simples (sem realizar seleção sobre os sinais disponíveis) varia entre 10 a 1.000 nanosegundos. Com um pós-processamento, a acuidade varia de 1 a 300 nanonsegundos.

3. Objetivos e Estado Atual

O presente trabalho visa o projeto de um serviço de sincronização de relógios para o sistema operacional QNX. Tal serviço procurará sincronizar os relógios das máquinas do sistema distribuído em relação ao tempo fornecido por um receptor de tempo GPS. Os protocolos a ser escolhidos terão de ter a capacidade de degradar para um estado de sincronização interna quando não for possível a obtenção do tempo a partir do receptor.

Deverão ser escolhidos e implementados dois protocolos de sincronização, sendo um determinístico e outro probabilístico. Como a rede de comunicação utilizada será Ethernet, o protocolo determinístico precisará de um anel lógico com passagem de *token* para o atraso máximo de viagem de uma mensagem seja conhecido. Esta medida trará, muito provavelmente, a perda de alguns serviços existentes no QNX, visto que todas as aplicações deverão utilizar o anel lógico e as que não usarem-no não poderão permanecer ativas. A possibilidade de que o sistema operacional continue funcionando sem tais serviços deverá ser analisada. A substituição de tais serviços por outros que utilizem o anel lógico também terá sua viabilidade estudada.

Durante a inicialização, o servidor de sincronização poderá receber como parâmetros o tipo de sincronização desejada, assim como o período máximo de permanência do *token* com cada participante do sistema. A partir disto, os sistemas cujas atividades forem baseadas no tempo poderão dispor do serviço.

A validação da proposta, especialmente no que diz respeito à acuidade [JAL94] (diferença de um relógio qualquer ao tempo oficial) obtida por cada um dos protocolos de sincronização, dependerá da análise das características do sistema operacional e da arquitetura de rede (com e sem anel). Isso se deve ao fato de que a medição dos resultados é um problema equivalente ao que se está tentando solucionar com o serviço de sincronização. Algumas aplicações podem vir a serem usadas também para verificar a utilidade do servidor.

Espera-se, ao final, construir um serviço de sincronização composto de software servidor e receptor GPS (incluindo antena e cabos) que possa ser utilizado em qualquer sistema distribuído com sistema operacional QNX instalado, utilizando rede Ethernet para comunicação de dados. A possibilidade de não aquisição do receptor GPS, caso concretizada, não acarretará na inviabilidade do trabalho, visto que o mesmo pode ser simulado.

4. Conclusões

A disponibilidade e acuidade do tempo fornecido pela constelação GPS, juntamente com o aprimoramento e a popularização dos receptores dos seus sinais, indicam esta forma de difusão de tempo oficial como a mais indicada para o uso em diversos tipos de aplicações. O uso do sistema operacional QNX sobre uma rede Ethernet, contudo, pode dificultar ou mesmo inviabilizar a construção de um protocolo de sincronização determinística. Caso o segundo caso venha a ocorrer, o sistema QNX/Ethernet não poderá manter um serviço de sincronização para as necessidades de aplicações críticas.

O serviço, ao ser usado com o protocolo determinístico e em redes locais distintas, procurará se adequar ao modelo CesiumSpray [VER97]. Neste modelo, redes locais acabam por se encontrar sincronizadas entre si, mesmo sem contato entre elas. Portanto, a sincronização se dará não somente dentro de LANs separadas, mas em WANs que disporem do serviço.

No futuro, o serviço poderá ser melhorado e ampliado (talvez técnicas de sincronização de relógios que se utilizem de novos paradigmas) ou usado em projetos de desenvolvimento de sistemas de tempo real e construção de outros serviços tolerantes a falhas.

5. Bibliografia

[AGN97] AGNOLETTO, Alessandro Dario. Sincronização de Relógios em Sistemas Distribuídos. Trabalho Individual, CPGCC - UFRGS, 1997.

[BIR96] BIRMAN, Kenneth P. Building Secure and Reliable Network Applications. Manning Publications, U.S.A., 1996.

[CRI89] CRISTIAN, Flaviu. Probabilistic Clock Synchronization. Distributed Computing, No. 3, Springer-Verlag, U.S.A., 1989.

[DAN97] DANA, Peter H. Global Positioning System (GPS) Time Dissemination for Real-Time Applications. Real-Time Systems, N. 12, Kluwer Academic Publishers, The Netherlands, 1997.

[JAL94] JALOTE, Pankaj. Fault Tolerance in Distribute Systems. Prentice Hall, U.S.A., 1994.

[LAM85] LAMPORT, Leslie, e SMITH, P. M. Melliar-. Synchronizing Clocks in the Presence of Faults. Journal of the ACM, Vol. 32, No. 1, U.S.A., 1985.

[QNX93] QNX Software Systems. QNX: System Architecture. Ontario, QNX Software Systems, 1993.

[TAN96] TANENBAUM, Andrew S. Computer Networks. Prentice Hall, U.S.A., 1996.

[VER97] VERISSIMO, P., RODRIGUES, L. e CASIMIRO A. CesiumSpray: a Precise and Accurate Global Time Service for Large-scale Systems. Real-Time Systems, N. 12, Kluwer Academic Publishers, The Netherlands, 1997.

Construção de Ferramentas de Comunicação de Grupo em Sistemas Tempo Real

Rafael Campello
campello@inf.ufrgs.br

Taisy Weber
taisy@inf.ufrgs.br

João Netto
netto@inf.ufrgs.br

Curso de Pós-Graduação em Ciência da Computação
Instituto de Informática
Universidade Federal do Rio Grande do Sul

Resumo

A comunicação de grupo tem sido um dos principais paradigmas de comunicação em sistemas distribuídos e muitas ferramentas têm sido construídas. Este trabalho apresenta as questões de projeto que devem ser levadas em consideração na construção de tais ferramentas para aplicações tempo real. A análise é baseada na experiência com um serviço de comunicação de grupo, chamado *RealGroup*, recentemente desenvolvido.

Abstract

The group communication has been one of the main communication paradigms in distributed systems and many tools have been built. This work presents the design issues that must be addressed in the building of those tools for real time applications. The analysis is based on the experience with a group communication service, called *RealGroup*, recently developed

1. INTRODUÇÃO

No desenvolvimento de sistemas distribuídos, um dos pontos mais positivos é o emprego de técnicas de tolerância a falhas. Com custos geralmente altos, devido às exigências de redundância, essas técnicas visam manter um fornecimento contínuo de serviços, mesmo na presença de falhas. Em ambientes distribuídos, entretanto, esse custo adicional é reduzido pela existência de elementos naturalmente redundantes e independentes. Por outro lado, o fraco acoplamento existente nesses sistemas, onde a única forma de interação é a troca de mensagens, gera uma maior dificuldade de desenvolvimento, agravada pelo paralelismo real na execução das diferentes partes do sistema.

A despeito de estilos de computação distribuída bem conhecidos como RPC e cliente/servidor, a comunicação de grupo vem ganhando grande aceitação. Mesmo úteis e maduros, esses estilos conhecidos mostram que sua natureza ponto-a-ponto e requisição-resposta não oferecem uma solução universal ao aumento da demanda de desenvolvimento de aplicações distribuídas. Nessas aplicações, interações mais complexas e altamente concorrentes entre os vários participantes são necessárias, reforçando as vantagens de um paradigma como a comunicação de grupo.

Além dos obstáculos naturais na implementação de uma ferramenta de comunicação de grupo, novas variantes surgem quando essa comunicação visa ser efetuada em sistemas distribuídos tempo

real. Nesses ambientes, fortes restrições temporais são impostas à comunicação, exigindo um projeto diferenciado e condizente com os novos cenários de falha possíveis.

Outra grande característica de sistemas computacionais aplicados à ambientes tempo real é a freqüente utilização de soluções proprietárias. Tanto em *software* como em *hardware* são usados elementos especificamente criados para cada aplicação, dada a necessidade de características não encontradas em componentes padronizados ou devido ao caráter demasidamente crítico de tais aplicações. Isso acaba agregando, a determinados projetos, custos muito elevados.

O Grupo de Tolerância a Falhas da UFRGS está desenvolvendo uma ferramenta para comunicação de grupo em sistemas distribuídos, voltada para o trabalho em ambientes tempo real, chamada *RealGroup*. Essa ferramenta usa *hardware* padrão, com computadores pessoais convencionais interligados por uma rede privada (ninguém mais tem acesso ao canal de comunicação) padrão Ethernet e não duplicada, além de um sistema operacional comercial, chamado QNX [QNX93].

Serão apresentados, na segunda seção deste artigo, detalhes sobre comunicação de grupo tempo real. Na seção 3 serão mostradas algumas diretrizes de projeto para ferramentas de comunicação de grupo em sistemas tempo real. Finalmente, serão apresentadas as conclusões e perspectivas futuras.

2. COMUNICAÇÃO DE GRUPO TEMPO REAL

A comunicação entre grupos de processos em ambientes distribuídos possui vários pontos importantes a serem analisados, incluindo o envio confiável de mensagens, a manutenção da consistência nas visões de grupo e a detecção e tratamento de falhas de processo e de comunicação [ABD96]. Em aplicações tempo-real distribuídas, que operam sob restrições de tempo e de confiabilidade, o problema torna-se mais complexo, já que atividades de difusão confiável e gerência de grupos devem ser realizadas em limites de tempo restritos, mesmo na presença de falhas. Nesse caso, cada processo membro precisa validar mensagens de difusão recebidas, dados seus prazos temporais, assim como a gerência de grupos deve garantir limites máximos de tempo no caso de falha ou recuperação de membros, ou em falhas na comunicação.

Outro aspecto importante em sistemas tempo-real é o seu sincronismo. Os sistemas distribuídos podem ser separados em duas correntes principais: síncronos e assíncronos. Uma definição comumente aceita de sincronismo é descrita pelas seguintes propriedades: velocidade de processamento limitada e conhecida; atrasos limitados e conhecidos na entrega de mensagens; taxa de desvio do relógio local limitada e conhecida. Com relação a protocolos de difusão para uso em sistemas tempo-real, uma discussão sobre a validade ou não de protocolos baseados em relógios lógicos (*timer-driven* ou *clock-less*) em aplicações tempo-real é apresentada por Veríssimo [VER90]. No entanto, para garantir um alto nível de sincronismo no sistema, serviços de sincronização de relógios são utilizados na maioria dos protocolos de difusão confiável tempo-real.

Vários protocolos tolerantes a falhas para difusão atômica e ordenada e para gerência de grupos foram propostos, tanto para sistemas assíncronos [CHA84] [MOS96] [REN96] [KAA92] como para sistemas síncronos [ABD96] [CRI85] [VER89] [KOP94].

O principal problema encontrado na implementação de protocolos de difusão confiável, gerência de grupos e sincronização de relógios, em sistemas tempo-real distribuídos, é a obtenção das propriedades de sincronismo descritas acima, principalmente valendo-se de componentes padronizados. Embora o determinismo de processamento seja, de certa forma, garantido pelas características de um sistema operacional tempo real, o determinismo em relação à comunicação é

um ponto muito delicado. Como toda a característica síncrona do sistema e dos protocolos está baseada nesse determinismo, é importantíssima a observação dessas propriedades. As características fundamentais para uma comunicação tempo-real, segundo Veríssimo [VER93], são: atrasos conhecidos e limitados na entrega de mensagens; comportamento determinístico na presença de distúrbios (sobrecargas, falhas); diferentes classes de latência, isto é, possível estabelecimento de prioridades à nível de mensagens; e finalmente conectividade.

Alguns protocolos padronizados de acesso ao meio tratam distúrbios, como omissões e sobrecargas, assim como implementam mecanismos de prioridades e tratam problemas de conectividade. São redes que oferecem serviços isócronos e de alta disponibilidade, valendo-se, muitas vezes, de algum tipo de redundância espacial. Um bom exemplo disso é a FDDI (*Fiber Distributed Data Interface*) [TAN96] [CHE97]. Com protocolos de acesso probabilístico ao meio, como é o caso das distintas variedades de CSMA (*Carrier Sense Multiple Access*), surgem problemas com o indeterminismo no atraso máximo de entrega das mensagens e na presença de distúrbios, a falta de classes de latência e a conectividade. Em qualquer ferramenta baseada nesse tipo de rede, é fundamental criar algum mecanismo de controle de acesso ao meio, eliminando as colisões e obtendo atrasos limitados. Outras características como conectividade e comportamento determinístico na presença de falhas só poderão ser alcançadas com o auxílio de algum tipo de redundância física do meio, aumentando os custos e prendendo a ferramenta a uma plataforma padronizada.

3. QUESTÕES DE PROJETO EM FERRAMENTAS DE COMUNICAÇÃO DE GRUPO TEMPO REAL

Como observado no desenvolvimento do *RealGroup*, a determinação dos critérios de projeto de uma ferramenta de comunicação de grupo passa pela idealização de um serviço voltado ao tipo de aplicação escolhida, pela determinação dos recursos disponíveis e, então, baseado em necessidades concretas, acaba na escolha de determinado protocolo de difusão e na adequação desses critérios às características desse protocolo.

Tendo em vista atender a demanda de sistemas com restrições temporais, é notória a importância em garantir todas as características de uma comunicação tempo real, apresentando um comportamento confiável e temporal. Características como atrasos limitados na entrega das mensagens, comportamento determinístico na presença de distúrbios e previsão de classes de prioridades são de extrema importância em face ao tipo de aplicação esperada. A rigidez com que essas características serão atendidas depende, basicamente, do tipo de aplicação esperada. Além disso, a possibilidade do envio tanto de mensagens esporádicas como de mensagens periódicas torna-se de grande valia para esse tipo de aplicação.

Outro critério importante a ser determinado é o tipo de plataforma a ser empregada. Como já mencionado, é comum o uso de plataformas proprietárias, com elementos de *hardware* duplicados, facilitando e garantindo a manutenção das características temporais exigidas. Com vantagens em relação a custos e portabilidade, entretanto, a utilização de uma plataforma padronizada, com elementos de grande aceitação no mercado, é outra alternativa possível. Essa escolha deve ser feita com base nas aplicações alvo dessa ferramenta. Baseada em componentes padronizados, por exemplo, não serão esperados sistemas de caráter extremamente crítico sobre tal serviço, requisitos normalmente satisfeitos por soluções proprietárias.

Por fim, detalhes sobre o protocolo de comunicação de grupo a ser usado surgem. Mesmo aparentando pouca relação com o tipo de aplicação prevista para o serviço, certas decisões não

podem deixar de lado o caráter temporal de tais aplicações. Decisões como semânticas de envio e de resposta, por exemplo, precisam levar em consideração a existência de limites temporais a serem cumpridos, inviabilizando qualquer comportamento não determinístico. As seguintes diretrizes precisam ser observadas:

- **Endereçamento**: mesmo sendo esse um critério independente das características tempo real a serem atendidas, é importante dar ao usuário de nível superior facilidades quanto à transparência no endereçamento de membros. Tendo isso em vista, o tipo de endereçamento ideal seria o via *kernel*. Assim, toda vez que um membro do grupo solicita o envio de uma mensagem para os outros, ele passa a mensagem para a camada de nível inferior (serviço de difusão), e este, de posse de uma lista de membros, difunde à todos. A maneira pela qual o processo gerenciador dissemina a mensagem está diretamente ligada à interface de rede existente, sendo tratada como detalhe de implementação. Com esse tipo de endereçamento, um membro pode disseminar uma mensagem para o resto do grupo sem saber quem são ou onde estão os membros destino.

- **Confiabilidade**: diretamente ligada ao tipo de aplicação alvo, a confiabilidade é um fator primordial na maioria dos sistemas tempo real. Assim, ferramentas que visem esses sistemas devem comportar-se de maneira confiável, mesmo em eventuais distúrbios como perdas ou duplicações de mensagens.

- **Semântica de falhas**: o tipo de falha a ser tolerado é outro critério importante. Diretamente ligada ao número de mensagens trocadas pelo protocolo, a semântica de falhas a ser usada geralmente tende a falhas por omissão. Essa escolha por suportar somente falhas por omissão, e não semânticas mais robustas, é justificada pelo alto custo das outras semânticas, associado à baixa ocorrência de outros tipos de falhas. Por outro lado, o suporte a essas outras semânticas pode ser importante, dependendo do tipo de aplicação alvo.

- **Ordenação**: Embora a ordenação total seja uma propriedade que, normalmente, eleva a quantidade de mensagens trocadas e a necessidade de processamento, é de relativa importância para a maior parte das aplicações distribuídas a observação dessa característica. Assim, é de grande interesse a garantia de uma ordenação total das mensagens enviadas pelo grupo, para que todos os membros do grupo recebam as mensagens sempre na mesma ordem.

- **Semântica de envio**: assim como o critério anterior, a semântica de envio a ser utilizada está fortemente relacionada com as necessidades da maioria das aplicações distribuídas. Para tanto, a garantia de uma entrega atômica das mensagens torna-se fundamental em muitos algoritmos distribuídos, principalmente em tarefas que envolvem a obtenção de concordância, sendo, também, ponto fundamental na maioria das ferramentas.

- **Semântica de resposta**: esse critério é limitado pelo comportamento temporal esperado. Semânticas que envolvem espera por respostas podem tornar indeterminado os atrasos possíveis, ou, quando *timeouts* são usados, agravar os cenários de pior caso. Por outro lado, a inexistência de respostas pode acarretar perda de confiabilidade pelo serviço. Assim, esse é um critério que deverá ser analisado com muito cuidado, devendo ser pesado bem na escolha do protocolo de difusão a ser utilizado.

- **Estrutura de grupos**: tratando de estrutura de grupos, quanto mais flexíveis e dinâmicas forem suas características, mais versátil às aplicações de nível superior o serviço será. Para tanto, grupos abertos, não-hierárquicos, com gerenciamento distribuído e com possibilidade de

sobreposição são as escolhas mais sensatas. Esses critérios estão, também, diretamente relacionados com o protocolo de difusão a ser utilizado.

Sendo esses os detalhes a serem observados, resta definir, baseado na realidade de recursos disponíveis ao serviço, quais seriam as maiores dificuldades na efetivação da ferramenta, adequando a mesma à essa realidade. No caso da ferramenta *RealGroup*, todo o sistema está colocado sobre uma plataforma de *hardware* padronizada, com interfaces, topologias e protocolos de rede existentes e comuns no mercado atual. Isso dificultou a efetivação do serviço, gerando a necessidade de adaptações para que o projeto inicial fosse viável, principalmente pela necessidade de um serviço síncrono, com atrasos determinísticos tanto quanto à entrega de mensagens como quanto ao tratamento de distúrbios.

4. CONCLUSÃO

Com base na peculiar forma de interação com o ambiente, apresentada pelos sistemas tempo real, pode-se afirmar que qualquer algoritmo ou mecanismo a ser utilizado nesse tipo de sistema precisa estar adequado às restrições temporais impostas. Teoricamente, como mencionado, esses limites temporais são razoavelmente claros, passando pela utilização de um sistema de comunicação tempo real, com características síncronas, e estendendo-se até o uso de protocolos tempo real, com tempos de latência determinísticos.

No nível de comunicação, técnicas que visam uma troca de mensagens confiável, com mecanismos que tratam a ocorrência de distúrbios, não fogem a essa regra, exigindo canais de comunicação com comportamento determinístico e protocolos que tenham tempos máximos de execução limitados. Usar recursos computacionais que atendam esses limites, como um sistema de comunicação com componentes replicados, por exemplo, bem como *softwares* básicos projetados especificamente para determinada aplicação, como um sistema operacional proprietário, são soluções eficientes e muito utilizadas em aplicações altamente críticas. Por outro lado, valendo-se de recursos computacionais padronizados, de grande aceitação e utilização no mercado atual, assim como de um sistema operacional não proprietário, buscar uma solução alternativa que obedeça os mesmos limites teóricos pode representar ganhos importantes no custo, principalmente para aplicações de caráter menos crítico.

5. REFERÊNCIAS

[ABD96] ABDELZAHER, Tarek et al. **RTCAST**. lightweight multicast for real-time process groups. [S.l.:s.n.], 1996. Disponível por FTP anônimo em ftp.eecb.umich.edu.

[CHA84] CHANG. J.; MAXEMCHUK, N. F. Reliable broadcast protocols. **ACM Transactions on Computer Systems**, New York, v.2, n.3, p.251-273, Aug. 1984

[CHE97] CHEN, Biao; KAMAT, Sanjay; ZHAO, Wei. Fault-tolerant, real-time communication in FDDI-based networks. **Computer**, Los Alamitos, v.30, n.4, p.83-90, Apr. 1997.

[CRI85] CRISTIAN, F. et al. Atomic broadcast: from simple message diffusion to byzantine agreement. In: 15th International Symposium on Falt Tolerant Computing Systems, 1985, Ann Arbor, USA. **Proceedings**. [S.l.:s.n.], 1985.

[KAA 92] KAASHOEK, Frans; TANENBAUM, Andrew. **Efficient reliable group communication for distributed systems**. Amsterdan: Department of Mathematics and Computer Science of Vrije Universiteit, 1992.

[KOP 94] KOPETZ, Hermann; GRÜNSTEIDL, Günter. TTP-a protocol for fault-tolerant real-time systems. **Computer**, Los Alamitos, v.27, n.1, p.14-23, Jan. 1994.

[MOS 96] MOSER, L. et al. Totem: a fault-tolerant multicast group communication system. **Communications of the ACM**, New York, v.39, n.4, p.54-63, Apr. 1996.

[QNX 93] QNX SOFTWARE SYSTEMS. **QNX**: system architecture. Ontario: QNX Software Systems, 1993.

[REN 96] RENESSE, Robert van; BIRMAN, Kenneth P.; MAFFEIS, Silvano. Horus: a flexible group communication system. **Communications of the ACM**, New York, v.39, n.4, p.76-83, Apr. 1996.

[TAN 96] TANENBAUM, Andrew. **Computer networks**. 3. ed. Upper Saddle River: Prentice-Hall, 1996.

[VER 89] VERÍSSIMO, P.; RODRIGUES, L.; BAPTISTA, M. AMp: a highly parallel atomic multicast protocol. In: SIGCOMM, 1989, Arlington, Texas. **Proceedings...** New York:ACM, 1989. p.83-93.

[VER90] VERÍSSIMO. P. Real-Time Data Management with Clock-Less Reliable Broadcast Protocols. In: Workshop on the Management of Replicated Data, 1990, Houston. **Proceedings**. New York: IEEE, 1990.

[VER 93] VERÍSSIMO, Paulo. Real-time communication. In: MULLENDER, S.J. (Ed.). **Distributed systems**. 2. ed. New York: ACM-Press, 1993. p.447-490.

Sistema de Controle de Trens - Desenvolvimento de uma Aplicação Simulada para Tolerância a Falhas

Marcelo André Minghelli
Fernanda Kruel Denardin
Ingrid E. S. Jansch-Pôrto
{mingheli, fernanda, ingrid }@inf.ufrgs.br

Instituto de Informática
Universidade Federal do Rio Grande do Sul

Resumo

Este artigo apresenta um tópico experimental em desenvolvimento no projeto de "sistemas distribuídos de alta confiabilidade para aplicações críticas". O aspecto abordado constitui-se em um enfoque preliminar da aplicação que será utilizada para testes das técnicas e ferramentas estudadas e desenvolvidas no decorrer do projeto.

Como aplicação básica optou-se por trabalhar com um sistema de controle de trens, que na vida prática enquadra-se na classe de aplicações críticas, atuando no transporte de pessoas ou cargas, onde qualquer tipo de perda ou acidente é totalmente indesejável.

Será apresentado como o sistema foi projetado e desenvolvido até o presente momento e o estado atual de desenvolvimento da simulação deste.

Abstract

This paper presents an experimental topic which is being developed in the context of the "high reliability distributed systems for critical applications project". The chosen aspect is a preliminary approach of the envisaged application that will be used to verify the techniques and tools studied and developed during the project.

As a basic application we have chosen to work with a train control system, which in fact is one of the critical applications, once they transport passengers or cargo and any kind of loss is completely undesirable.

In this paper, we will present how this system has been designed and developed to this moment, and the actual development state of this system simulation.

1- Introdução

Alta confiabilidade e disponibilidade sempre foram requisitos essenciais para uma classe de sistemas computacionais classificados como sistemas críticos. Esses sistemas são os utilizados em situações onde eventuais falhas podem por em risco vidas humanas (centrais nucleares, transporte, controle de indústrias, medicina) ou provocar grandes danos financeiros (transações bancárias e bolsas de valores).

Atualmente, um dos projetos de pesquisa do grupo de tolerância a falhas da UFRGS[1] é o projeto SisDAC - Sistemas Distribuídos de Alta Confiabilidade para aplicações críticas

[1] Maiores informações podem ser obtidas na página do grupo de tolerância a falhas em http://www.inf.ufrgs.br/gpesquisa/tf/.

[JAN97] . O principal objetivo deste projeto é justamente aplicar técnicas de tolerância a falhas a sistemas distribuídos utilizados em situações críticas, visando alcançar um grau elevado de confiabilidade e disponibilidade em tais sistemas.

Para isso, um dos trabalhos em andamento é dedicado a preparar o suporte para o desenvolvimento e a construção de uma aplicação com características de funcionamento ideais para testes do projeto. A idéia inicial era de montar um sistema de controle de trens físicos baseados em modelos HO, entretanto diante das dificuldades enfrentadas para especificar e adquirir partes no mercado nacional, optou-se por, inicialmente, simular este sistema, conforme será apresentado neste artigo. Cabe ressaltar que a idéia de se trabalhar com trens não é original; um trabalho tomando por base este modelo foi desenvolvido nacionalmente no grupo de ciência da computação da UNICAMP por Elbson Quadros e Cecília Rubira [QUA97].

2- O Sistema de Controle de Trens

O projeto SisDAC prevê o desenvolvimento e a integração de métodos e técnicas de tolerância a falhas aplicáveis a sistemas de controle críticos. A viabilidade das soluções propostas deve ser visualizada através do uso de um sistema de controle automático de trens simulado. Um dos pontos importantes da definição deste protótipo é que todos os componentes da ferrovia ou do trem devem ser tolerantes a falhas.

O funcionamento geral do sistema supõe que o conjunto de trilhos será dividido em malhas, onde cada uma possuirá uma unidade de controle própria. A interação entre estas unidades de contole é estabelecida de forma a comportar-se de acordo com as propriedades dos sistemas distribuídos, permitindo a comunicação de todas as partes e o maior controle organizado do sistema, além do usofruto das facilidades oferecidas pelo modelo. Também as dificuldades associadas ao modelo tais como ausência de memória global, ausência de relógio global e conhecimento limitado das informações referentes às outras malhas, são de interesse do estudo do caso-exemplo.

Será permitido ao usuário alterar fatores como velocidade, trajeto, número de trens na ferrovia e destino de cada um deles, antes do início de cada simulação.

3- A Ferramenta SIMOO MET

Para a implementação deste projeto optou-se por utilizar a ferramenta SIMOO [COP97], desenvolvida por Bernardo Copstein em sua tese de doutorado, no CPGCC (*Curso de Pós-Graduação em Ciência da Computação*) da UFRGS. Esta ferramenta constitui-se em *framework* para simulação discreta orientada a objetos de propósitos gerais, que utiliza um diagrama de classes hierárquico enriquecido com recursos para a construção de modelos de simulação. Uma grande vantagem no uso desta ferramenta é que ela encontra-se disponível, para uso, no Instituto de Informática.

A ferramenta possui uma biblioteca de elementos de visualização que permite a criação de um fundo para a simulação e a movimentação de imagens (tipo *bitmap*), adicionando um aspecto visual e gráfico ao sistema simulado.

Pelas constatações decorrentes do estudo inicial da ferramenta, acredita-se que a sua adoção, altamente motivada pelas facilidades de obtenção e fácil acesso ao seu autor, revelou-se positiva.

4- Apresentação do Sistema

4.1- Uma Visão Geral do Sistema

O foco do presente trabalho consiste na implementação e controle de um modelo de ferrovia utilizando um ambiente de simulação. O modelo inicial definido para a simulação possui um conjunto de trilhos e conectores e dois trens percorrendo a mesma ferrovia, mas em sentidos opostos. O objetivo de ambos os trens (que são identificados como *"Azul"* e *"Vermelho"*) é percorrer o trilho até chegar na estação de cor diferente e retornar à sua própria estação por um número determinado de vezes, sem que haja colisão entre eles, terminando por estacionar junto à sua estação após a última volta. Por definição, o trem *Azul* representa um trem de passageiros e o *Vermelho*, um trem de carga.

4.2- Estrutura do Sistema

Cada trem possui uma unidade de controle própria que gerencia a sua movimentação e faz uma análise de colisão. Quando é estabelecido o processo de simulação, o relógio de simulação é inicializado e são criados os elementos gráficos de fundo, juntamente com os dois trens, o que permite a visualização da simulação. A partir daí, o controle central envia uma mensagem para cada controle de trem que coordena independentemente o melhor trajeto durante uma volta. Durante cada volta, o controle realiza a movimentação no setor, a escolha do próximo setor, a escolha do setor a ser colocado na fila de previsão (que será detalhado adiante) e faz o deslocamento dentro da fila. Cada trecho de trilho em reta ou curva corresponde a um setor. Estas rotinas são executadas até que se complete uma volta.

Com relação a tolerância a falhas, o sistema só foi programado para possíveis falhas de controle que causem colisão entre os dois trens. Estas falhas de controle podem colocar dois trens em sentidos opostos em um mesmo setor de trilho ou colocar um trem em uma rota indefinida, fazendo com que a simulação não termine. Como o sistema atual constitui-se em uma versão inicial de teste, ainda não foi implementada a simulação de falhas nos trilhos, conectores e trens, ou seja, os elementos correspondentes às falhas físicas de um modelo real.

4.3- Lógica de Movimentação

A movimentação dos trens é baseada na biblioteca de elementos gráficos de visualização do SIMOO. Há duas rotinas de movimentação por setor de trilhos: uma para curvas e outra para retas. Para retas, o processo é bastante simples: a rotina reta é ativada por uma mensagem e recebe como parâmetros o *bitmap* correspondente ao sentido da reta, a posição inicial em coordenadas cartesianas em relação ao *bitmap* de fundo (x e y) e o valor dos incrementos de x e y em cada movimentação.

Para a movimentação em curvas, o processo é um pouco mais complicado devido às limitações do simulador. Como o simulador não rotaciona os *bitmaps*, é necessário que se criem tantos *bitmaps* quantos forem necessários para dar a impressão de movimento na curva. Neste modelo foram necessários nove *bitmaps* para uma representação real. Assim, para cada curva devem ser indicados os nomes e as coordenadas dos nove *bitmaps* correspondentes. Para dar a impressão de movimentação, os *bitmaps* são criados e apagados, funcionando como quadros em uma animação.

4.4- A Previsão de colisão

A previsão de colisão utiliza um método relativamente simples. Como cada setor de trilho e cada conector são identificados, é possível determinar os próximos movimentos do trem. Para evitar uma colisão com um trem em sentido oposto, é necessário saber se um determinado setor de trilho vai ser utilizado pelo outro trem, antes de completar-se a movimentação. Para isso, cada controlador de trem possui uma fila contendo a identificação do setor atual, o próximo setor e uma previsão do movimento seguinte. Como há um trajeto que deve ser realizado, mas não é possível que ambos os trens utilizem simultaneamente o mesmo setor, a cada movimentação é feita uma comparação entre os elementos da fila.

Na figura 1, é representado um modelo utilizado na simulação. Há um trilho composto de onze setores de trilho (1 a 11), sendo oito em curva e três em reta, seis conectores(A até F) e os trens *Vermelho* e *Azul*. Nesta figura, são mostradas duas situações na simulação: com ambos os trens na sua posição inicial e com ambos os trens no setor dez após duas movimentações.

Analisando a figura 1, verifica-se que os dois trens, se programados para realizar o movimento básico, devem utilizar o setor dez simultaneamente na terceira movimentação. Neste caso, essa colisão já poderia ser prevista e evitada no início da volta. Mas um dos dois deve ter a preferência na passagem. Como o trem *Azul* transporta passageiros, supõe-se ser prioritário o conforto destes mediante a redução do tempo de viagem; logo, ele terá a preferência de passagem. No início da volta do trem *Vermelho*, a previsão de colisão indicaria o setor dez. Então, é ativada uma rotina para selecionar uma rota alternativa, até que ele possa utilizar o setor dez, com base na otimização dos parâmetros tempo e distância. Assim, o elemento de previsão na fila é alterado e, então, é executada a movimentação dos dois trens.

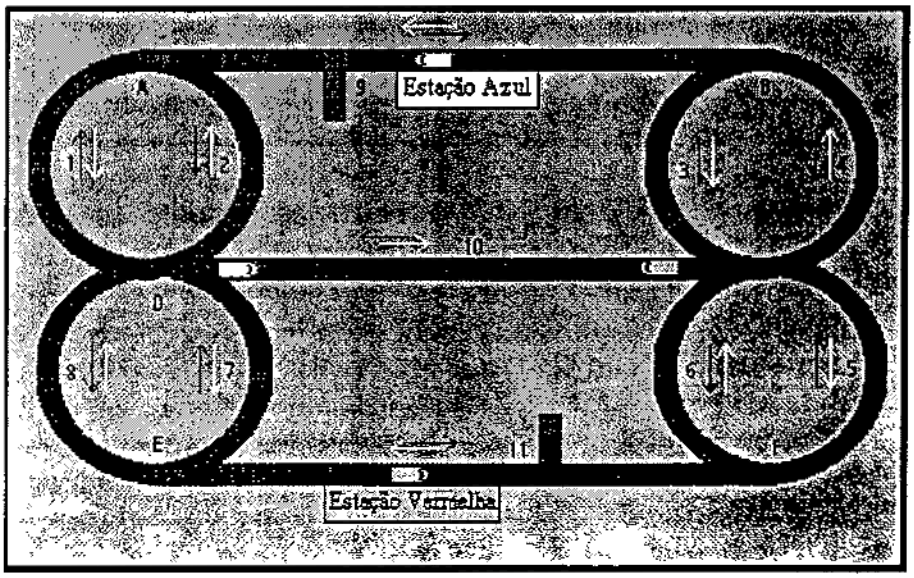


FIGURA 1 - Visualização Grafica do Sistema de Controle de Trens

5- Conclusão e Extensões Futuras

Este modelo simplificado constitui-se apenas em uma experiência inicial, se considerados os objetivos do projeto. Há várias etapas a serem cumpridas, se for considerado o que o projeto necessita para subsidiar os testes das demais técnicas de tolerância a falhas que serão estudadas.

Algumas perspectivas de trabalhos futuros, que podem ser incorporados em pouco tempo, referem-se à extensão da malha ferroviária, bem como o número de estações e de trens trafegando. Além disso, seria interessante a simulação abrir a possibilidade de se injetar falhas, a cada período de tempo, em algum setor de trilho ou em conectores com defeito, já que em um modelo real, estes elementos não são à prova de falhas.

Também deve ser implementado um controle auxiliar para que o usuário possa acessar o sistema, em tempo de execução, e fazer alterações na rota, velocidade ou destino de um determinado trem.

Até o presente momento, este trabalho teve por objetivos principais a exploração e a identificação dos recursos de uma ferramenta nova, o SIMOO, além do aprendizado das características de um novo tipo de aplicação, do ponto de vista dos autores. A partir destes elementos, torna-se viável a especificação de um novo ambiente com características e potencialidade adequadas ao projeto no qual se enquadra. Os resultados decorrentes deste permitirão observar o comportamento do sistema físico correspondente associado a todas as facilidades de atualização e alteração proporcionadas pelos sistemas computacionais.

6 - Créditos

Os autores Fernanda e Marcelo devem sua participação neste projeto ao financiamento de bolsas DTI e Iniciação Científica, respectivamente, no contexto do programa RHAE do CNPq.

7- Referências Bibliográficas

[COP97] COPSTEIN, Bernardo. **SIMOO: Plataforma Orientada a Objetos para Simulação Discreta Multi-Paradigma.** Tese de Doutorado. Porto Alegre: CPGCC da UFRGS, 1997.

[QUA97] QUADROS, Elbson; RUBIRA, Cecília. **Uma experiência prática em programação orientada a objetos, distribuída e confiável.** In: Simpósio de Computadores Tolerantes a Falhas, 7. Anais. Campina Grande: SBC, 1997.

[JAN97] JANSCH-PÔRTO, Ingrid; WEBER, Taisy; WEBER, Raul. **SisDAC - Sistemas Distribuidos de Alta Confiabilidade para aplicações críticas.** Projeto submetido e aprovado pelo programa RHAE/CNPq. Em execução desde 1997.

Segurança em Sistemas de Micropagamentos Eletrônicos

Alexandre M. Braga Delano M. Beder Ricardo Dahab
Cecília M. F. Rubira

Universidade Estadual de Campinas
Instituto de Computação
Caixa Postal 6176
13081-970 Campinas SP
Fone/fax:+55+19+2393115
e-mail:{972314,delano,rdahab,cmrubira}@dcc.unicamp.br

Sumário

A criptografia é usada para proporcionar segurança em aplicações de comércio eletrônico na Internet e na geração de dinheiro eletrônico destas aplicações, em particular, nos sistemas de micropagamentos eletrônicos: pagamentos de valor muito baixo feitos muito rapidamente e a freqüências altas. No comércio eletrônico com distribuição *on-line* de produtos, transferências de valores e de produtos devem ser tratadas como transações atômicas. Nestes sistemas, mecanismos de recuperação de erros por avanço e por retrocesso são combinados para prevenir fraudes. *PayPerClick* é um sistema para venda e distribuição *on-line* de publicações na Internet baseado em micropagamentos.

Palavras-chave: criptografia, comércio eletrônico, micropagamento, orientação a objetos.

Abstract

Cryptography is used to offer security and also to generate electronic currency for electronic commerce applications over the Internet, especially for those using micropayment schemes. In such schemes, payments involving very low amounts are done quickly and frequently. An important aspect of electronic commerce systems with on-line distribution of goods is that value and goods transferences must be atomic transactions. In such systems, forward and backward error recovery are combined to prevent fraud. *PayPerClick* is a tool for electronic sale and on-line distribution of publications based on micropayments.

Key words: cryptography, electronic commerce, micropayment, object orientation.

1 Introdução

A tecnologia para pagamentos eletrônicos seguros pela Internet foi herdada da época em que a segurança de informações interessava somente aos militares, mas está sendo usada hoje para proporcionar segurança, anonimato e privacidade aos usuários de comércio eletrônico. A criptografia tem papel importante não somente na solução de problemas de segurança, mas também na geração de dinheiro eletrônico, em particular, nos micropagamentos eletrônicos [2, 3, 9, 12, 13]. O texto a seguir está organizado da seguinte forma. A Seção 2 trata dos aspectos criptográficos e transações críticas do pagamento eletrônico. Na Seção 3, os detalhes da geração de moedas para micropagamentos são descritos. O *PayPerClick*, um *software* para venda e distribuição *on-line* de publicações na Internet, é descrito brevemente na seção 4. Conclusões e trabalhos futuros são apresentados na Seção 5.

2 Criptografia, Transações e Pagamento Eletrônico

Comércio eletrônico é qualquer forma de transação de negócios na qual as partes interagem eletronicamente, em oposição ao intercâmbio físico ou contato físico direto [16]. Três entidades estão associadas a uma transação comercial eletrônica: recebedor, pagador e, opcionalmente, uma instituição financeira intermediária. O pagamento eletrônico usado em comércio eletrônico pode ser classificado em quatro categorias [15]: dinheiro eletrônico, cheque eletrônico, transferência eletrônica de fundos e cartão de crédito.

No comércio eletrônico com pagamento eletrônico e distribuição *on-line* de produtos, compras envolvem duas transações críticas: transferência de produtos e transferência de valores. A primeira faz parte da segunda e ambas devem ser transações atômicas. O conceito de transação atômica originou-se de pesquisas em gerência de banco de dados. Atualmente, ambientes de programação distribuída transacionais têm reforçado a tese que motivou o uso de ações atômicas em ambientes distribuídos: ambientes transacionais oferecem uma abstração poderosa e eficiente para a programação de sistemas distribuídos tolerantes a falhas.

Transações atômicas [5] têm três propriedades que ajudam a diminuir a complexidade da programação distribuída: (*i*) seriação: garante que a execução de programas concorrentes que compartilham objetos é livre de interferência, isto é, uma execução concorrente é equivalente a alguma execução na qual acessos a objetos compartilhados ocorrem de forma serial. A implementação de um protocolo para o controle de concorrência é necessário a fim de garantir esta propriedade de transações atômicas. (*ii*) atomicidade: garante que uma transação atômica termina somente em um de dois estados: (*a*) normal, no qual é validada e produz o resultado desejado; (*b*) anormal, na qual não produz resultados porque foi abortada. Técnicas de recuperação de erros por retrocesso [11] são utilizadas na implementação dos mecanismos que garantem esta propriedade. Falhas típicas incluem a parada de uma estação de trabalho ou falhas de comunicação. (*iii*) permanência de efeito: garante que qualquer resultado produzido por uma transação atômica não será desfeito devido a falhas. É implementada através de um armazenamento dos resultados em memória estável. Um protocolo de controle de término da transação atômica (*commit protocol*) garante que todos os objetos modificados

dentro de uma transação atômica têm o seu estado escrito em memória estável, no caso de *commit*, ou que as modificações não são gravadas, no caso de aborto.

Transações atômicas fornecem um mecanismo de tolerância a falhas com recuperação de erros por retrocesso de estado, mas há aplicações que requerem também recuperação de erros por avanço de estado. Recuperação de erros por retrocesso retorna o sistema para um estado prévio livre de erros sem requerer nenhum conhecimento dos erros. Recuperação de erros por avanço (geralmente esquemas de tratamento de exceções) é baseada no uso de dados redundantes e repara o sistema através da análise do erro detectado e colocando-o num estado correto. Na atomicidade de transferência de produtos, o dinheiro eletrônico deve ser transferido atomicamente e esta ação deve obrigar à transferência do produto [6].

Técnicas criptográficas são usadas de dois modos em sistemas de pagamento eletrônico: (*i*) como serviços criptográficos primitivos na implementação dos protocolos de segurança; (*ii*) na adaptação de protocolos já existentes em aplicações de comércio eletrônico. Dois exemplos são os sistemas de micropagamentos baseados em cadeias de *hash* [13, 9, 3], usadas anteriormente para autenticação de *passwords* em canais de comunicação inseguros [10], e aqueles baseados em sistemas para controle de cotas de uso dos recursos em um sistema distribuído [12].

Sistemas de dinheiro eletrônico são divididos em duas categorias, de acordo com o compromisso entre segurança, desempenho e valor das transações: (*i*) os baseados em funções de *hash* e (*ii*) os baseados em assinaturas digitais. No primeiro caso, os sistemas são usados em micropagamentos e baseiam-se em cadeias de *hash* propostas por Lamport [10]. Alguns sistemas também usam assinaturas digitais e certificação digital de chave pública [13, 3, 9]. Estes sistemas não garantem anonimato e o gasto repetido da mesma moeda é inibido com moedas específicas do cliente, do vendedor ou da transação. Dos sistemas baseados em assinaturas digitais, aqueles baseados em assinaturas cegas garantem o anonimato e a privacidade do usuário [7, 14].

3 Micropagamentos

Micropagamentos são pagamentos de valor muito baixo feitos muito rapidamente [4] e a freqüências altas. Esquemas de micropagamentos com muitos pagamentos repetidos tratam um número muito grande de transações em intervalos de tempo relativamente pequenos e usam protocolos criptográficos de segurança simples e computacionalmente eficientes. Moedas eletrônicas são elementos de cadeias de *hash* gerados da seguinte forma: uma semente aleatória, x, é escolhida e a cadeia $A_0, A_1, \ldots, A_{n-1}$ é computada recursivamente, como segue:

$$A_0(x) = x$$
$$A_{i+1}(x) = h(A_i(x)) \text{ e } A_i \neq A_j \text{ para } i \neq j,$$

onde h é uma função de *hash* unidirecional e com baixíssima probabilidade de colisões.

As moedas $A_0, \ldots, A_{n-1}$ permitem até n micropagamentos de um valor fixo v já estabelecido. Antes de qualquer pagamento, o pagador envia ao recebedor A_n e v de modo autêntico. O pagador garante que A_n é de fato o extremo de uma cadeia de *hash* usada em

pagamentos subseqüentes. Os micropagamentos são efetuados pelo envio, na ordem inversa, $A_{n-1}, A_{n-2}, \ldots, A_0$, de elementos da cadeia para o recebedor. A verificação dos pagamentos é feita pela reconstituição parcial de elementos da cadeia a partir do pagamento anterior, A_{n-i}, até o atual A_{n-j}, $i < j$, como abaixo:

$$A_{n-i} = h_0(h_1(h_2(...h_{k-1}(A_{n-j})))), \text{ onde } k = j - i.$$

O valor do pagamento é o produto vk. A_{n-j} assume o papel do pagamento anterior na próxima verificação. O recebedor envia para a instituição financeira intermediária a cadeia $A_{n-i}, \ldots, A_{n-j}$ para ser reembolsado de kv em dinheiro real. Com esse esquema simplificado é possível inibir furtos. O pagador não pode gerar a mesma moeda duas vezes, porque a seqüência de moedas é verificada em relação ao pagamento anterior e deve ocorrer pelo menos uma iteração da função de *hash* sobre o pagamento atual. O recebedor não pode forjar moedas e receber mais que o devido, uma vez que não é capaz de gerar elementos da cadeia para diante a partir do registro do último pagamento. Por outro lado, um intruso tem a possibilidade de fraudar o sistema de três formas: (*a*) ganhando acesso aos registros temporários de pagamentos mantidos pelo vendedor, (*b*) interceptando a comunicação entre pagador e recebedor pelo monitoramento do canal de comunicação ou observando a execução do programa de verificação das moedas e (*c*) obtendo acesso ao sistema pela obtenção ou descobrimento da senha.

A terceira forma só pode ser eliminada com uma forma de identificação baseada em características físicas intrínsecas ao usuário. Fraudes a partir da leitura de registros dos pagamentos ou pela observação da verificação não são possíveis pelos mesmos motivos citados acima. Um intruso nas condições do segundo caso tem a possibilidade de fraudar se conseguir uma moeda cuja posição na cadeia esteja à frente daquela correntemente registrada pelo recebedor como último pagamento e tiver a oportunidade de gastá-la. Esta tal moeda será válida de acordo com as verificações. Esta situação é possível após uma queda do sistema seguida de uma recuperação de erros por retrocesso, considerando pagamentos como transações atômicas, do seguinte modo: o intruso está monitorando e copiando constantemente os pagamentos em trânsito de um usuário. Eventualmente, o sistema do recebedor cai antes que o pagamento novo seja verificado e registrado. Neste caso, ocorrendo uma recuperação de erros por retrocesso, o sistema volta para o estado estável anterior, mas o intruso possui uma cópia de uma moeda que, de acordo com os registros do vendedor, ainda não foi gasta.

Supondo que o intruso possa personificar o pagador, esta fraude pode ser inibida de duas formas. Primeira, solicitando ao pagador cuja transação de pagamento não foi completada uma cadeia nova de moedas. Segunda, tirando vantagem das propriedades de sincronização da comunicação com segurança baseada em cadeias de *hash* [10]. Por exemplo, o último pagamento registrado contém a moeda A_i e o pagamento interceptado pelo intruso contém a moeda A_{i-m}, para $m > 0$. No primeiro pagamento após a queda do sistema, daquele pagador cuja transação de pagamento não foi completada, o vendedor pode requisitar mais um pagamento compulsório. Assim, o recebedor receberá a moeda A_{i-x}, para $x \geq m$, e exigirá a moeda A_{i-x-1}, possuida somente pelo pagador verdadeiro. A fraude é inibida e a perda eventual da moeda A_{i-x-1} pode ser compensada por descontos em compras subseqüentes.

Estas soluções combinam os esquemas de recuperação de erros por avanço e retrocesso. Recuperação de erros por avanço do estado inibe fraudes nas transferências de valores em sistemas de micropagamentos porque moedas interceptadas por intrusos, antes de quedas do sistema, podem ser invalidadas mais facilmente que na recuperação por retrocesso. Mas a detecção de transações incompletas e a comunicação adicional podem ter custo computacional alto em relação ao prejuízo com a fraude. Se quedas forem raras e os valores dos pagamentos muito baixos, o sistema pode tolerar furtos pequenos e pouco freqüentes.

4 Exemplo de Aplicação: *PayPerClick*

Um *software* para venda e distribução *on-line* de publicações acessíveis a partir de *web browsers* e baseado em micropagamentos eletrônicos, o *PayPerClick*, foi implementado usando técnicas de orientação a objetos. Um esquema de micropagamento baseado em cadeias de *hash*, semelhante ao apresentado em [13], foi usado. O projeto apresenta uma arquitetura composta por quatro subsistemas: transações, porta-moedas eletrônicos, controladores de vendas e composições recursivas de hiperdocumentos. A linguagem Java foi usada com o JDK 1.1.2. O *framework* criptográfico Java (JCA) [8] foi usado nas operações de geração das moedas eletrônicas, assinaturas digitais e verificação dessas. O algoritmo de assinatura digital usado foi o DSA com chaves de 512 bits. As assinaturas digitais possuem apêndice. As moedas eletrônicas foram geradas pela função de *hash* SHA sem chave. O hotjava 1.1b2 foi o *web browser* usado e o recurso de *applets* assinadas, integrado ao *browser*, proporcionou a capacidade de verificação da integridade e autenticidade do código da *applet*, tornando-o confiável. O Java *Remote Method Invocation* (RMI) [1] foi usado na implementação da comunicação baseada no modelo cliente-servidor entre os módulos do consumidor e do vendedor da aplicação.

5 Conclusões e Trabalhos Futuros

O uso comercial da Internet faz surgir várias possibilidades novas de negócios, assim como contextos novos para os problemas de segurança de informações. Segurança baseada em criptografia, técnicas de tolerância a falhas e mecanismos de recuperação de erros são partes fundamentais no desenvolvimento de *software* para comércio eletrônico baseado em micropagamentos eletrônicos e distribuição *on-line* de produtos. De fato, o *software* para comércio eletrônico só possui algum valor com a aplicação correta e combinada destas técnicas. Tarefa nada trivial e muito suscetível a erros. Em particular, na implementação de porta-moedas eletrônicos, somente a correção do modelo matemático de geração das moedas não garante a segurança da aplicação. Os serviços criptográficos disponíveis, tais como funções de *hash* e algoritmos de assinatura digital, devem não somente ser bons, mas também implementados de forma segura. *Software* no qual a segurança é crítica deve ser capaz de escolher entre algoritmos e implementações diferentes baseando-se na disponibilidade de recursos, mas sem comprometer a qualidade dos serviços de segurança.

Referências

[1] Java Remote Method Invocation. http://java.sun.com/products/jdk/rmi/index.html.

[2] Netbill: an Internet Commerce System Optimized for Network Delivered Services. IEEE CompCon Conference, Março 1995. http://www.ini.cmu.edu:80/netbill/pubs.html.

[3] Ross Anderson, Charalampos Manifavas e Chris Sutherland. Netcard — A Practical Eletronic Cash Scheme. http://www.cl.cam.ac.uk/users/rja14/.

[4] N. Asokan, Philippe A. Janson, Michael Steiner e Michael Waidner. The State of the Art in Eletronic Payment Systems. *IEEE Computer*, páginas 28–35, Setembro 1997.

[5] P. A. Bernstein, V. Hadzilacos e N. Goodman. *Concurrency Control and Recovery in Database Systems*. Addisson-Wesley Publishing Company, 1987.

[6] L. Jean Camp, Marvin Sirbu e J. D. Tygar. Token and Notational Money in Electronic Commerce. *Usenix Workshop on Electronic Commerce*, New York, NY, Julho 1995.

[7] David Chaum. Security Without Identification: Card Computers to Make Big Brother Obsolete. *Communications of the ACM*, 28(10), Outubro 1985.

[8] Mary Degeforde. Java Cryptography Architecture API Specification and Reference. http://java.sun.com/products/JDK1.1/docs/guide/security/CryptoSpec.html, Fevereiro 1997.

[9] Steve Glassman, Mark Manasse, Martí Abadi, Paul Gauthier e Patrick Sobalvarro. The Millicent Protocol for Inexpensive Electronic Commerce. http://www.millicent.digital.com.

[10] Leslie Lamport. Password Authentication with Insecure Communication. *Communications of the ACM*, 24(11):770–772, Novembro 1981.

[11] P.A. Lee e T. Anderson. *Fault Tolerance: Principles and Practice*. Springer-Verlag, 2a. edição, 1990.

[12] B. Clifford Neuman e Gennady Medvinsky. Requirements for Network Payment: The Netcheque Perspective. *IEEE Compcon' 95*, San Francisco, Março 1995.

[13] Ronald L. Rivest e Adi Shamir. Payword and Micromint: Two Simple Micropayment Schemes. http://theory.lcs.mit.edu/rivest/~rivest/publications.html. Maio 1996.

[14] Bruce Schneier. *Applied Cryptography — Protocols. Algorithms . and Source Code in C*. John Wiley and Sons. 2a. edição. 1996.

[15] Kiyoon Sung e Jae Kyu Lee. Analysis and Design of the Internet Based Payment System. Dissertação de Mestrado. Department of Management Engineering - Graduate School of Management - Korea Advanced Institute of Science and Technology, 1996. Disponível em http://www.dcc.unicamp.br/~cripto/artigos/analysis.design.pay.system.html.

[16] Paul Timmers. Eletronic Commerce — An Introduction. http://www.cordis.lu/esprit/src/ecomint.html, Maio 1996.

Substituição Dinâmica de Classes com Validação de Objetos

Werner Haetinger
wernerh@inf.ufrgs.br

Maria Lúcia Blanck Lisbôa
llisboa@inf.ufrgs.br

Universidade Federal do Rio Grande do Sul
Instituto de Informática
Curso de Pós-Graduação em Ciência da Computação - CPGCC

Av Bento Gonçalves, 9500 - Bloco IV
CEP 91501-970 Porto Alegre - RS

Resumo

A necessidade de evolução de software é uma realidade presente em todos os sistemas de computação, seja para alterar ou adicionar funcionalidades. Um importante aspecto a considerar é a possibilidade de antecipar, ainda na fase de projeto, os tipos de alterações que o software poderá sofrer durante a sua fase operacional. A antecipação permite construir componentes de software que isolem os aspectos sujeitos a alterações, de forma a facilitar a sua substituição por uma nova versão. O sistema de substituição dinâmica aqui apresentado utiliza o modelo de objetos para a construção de componentes encapsulados e emprega reflexão computacional para hospedar técnicas de tolerância a falhas, visando assegurar a manutenção da confiabilidade da nova versão do software.

Abstract

Software evolution must be considered in the context of all computational systems. Evolution includes both correction and modifications to add new capabilities. It is important to anticipate the kinds of alterations to be carried on during the software lifetime in order to promote its maintainability. This permits to encapsulate the software components prone to alteration, thus facilitating the substitution of these components by new versions.

The dynamic modification system proposed is based on the object model to structure encapsulated components and uses computational reflection to host fault-tolerant techniques. Those techniques contribute to assure the reliability of the new software version.

1 Introdução

Depois que um sistema entrou em regime de operação normal e precisa sofrer uma alteração, a abordagem mais comum é descontinuar o programa que está em execução e então substituí-lo pela nova versão. O sistema já alterado é então reinicializado, passando a estar novamente disponível ao usuário. Esta interrupção ocasiona a indisponibilidade do serviço aos usuários do software, e, também, pode implicar em diminuição de confiabilidade, pela inclusão de novo código na aplicação.

Constata-se que há necessidade de novas técnicas para manutenção que não interrompam a operação do sistema por longos períodos. Uma destas técnicas é utilizar um sistema que faz a substituição dinâmica da versão de um programa, eliminando a etapa de desativação do sistema. A substituição é facilitada quando foram previstos, ainda na fase de projeto, os tipos de alterações que o software poderá sofrer durante a sua fase operacional.

A antecipação permite construir componentes de software que isolem os aspectos sujeitos a alterações, de forma a facilitar a sua substituição. Para atender a este aspecto, o modelo de objetos é particularmente adequado: permite confinar, encapsular, esconder e proteger estruturas de dados e correspondente código de manipulação, oferecendo seus serviços por meio de interfaces bem definidas e estáveis. A estabilidade das interfaces pode ser assegurada mesmo que sejam feitas alterações em qualquer das propriedades – dados ou código – do componente.

A evolução de um software é o tema central deste trabalho, cuja preocupações são a continuidade da disponibilidade dos serviços durante a transição de uma versão de software, que se encontra em operação, e a preservação da confiabilidade, na versão mais atualizada. Técnicas de tolerância a falhas são usadas para garantir a validação da substituição, ou seja, garantir que o novo componente é funcionalmente equivalente ao componente que foi substituído. O modelo de objetos é adotado para a construção de componentes encapsulados e técnicas de reflexão computacional são usadas para separar, em um meta-programa, as atividades relacionadas ao processo de substituição e validação [LIS97].

2 Validação e Testabilidade

A validação da substituição é o elemento inovador mais importante desta proposta, e exerce um papel similar ao da testabilidade de um programa. Os projetos pesquisados na literatura [FRZ97], [GUP93], [GUP96], [SEG93] abordam principalmente os aspectos ligados à carga e substituição dinâmica de módulos em si, mas não apresentam soluções para a detecção e recuperação de falhas durante o processo de substituição.

Os mecanismos de introspecção da reflexão computacional são bastante adequados para dar suporte à validação dos componentes. A abordagem mais simples é aplicar o teste da caixa preta sobre o novo componente durante a validação. Isto é feito através da comparação entre os resultados fornecidos pela versão antiga que executa juntamente com a versão nova. Caso os resultados sejam divergentes é assumido que a nova versão apresenta falha.

2.1 Técnicas de Tolerância a Falhas

As técnicas de tolerância a falhas - TF agregam significativo conhecimento sobre gerenciamento de componentes. Através destas técnicas o comportamento dos componentes envolvidos no processo de substituição pode ser monitorado: o objeto antigo que tem um

funcionamento correto e bem conhecido pelo sistema e o objeto novo que pode apresentar algum comportamento inesperado como reação a alguma mensagem.

A técnica de TF que está presente de forma mais marcante é blocos de recuperação [RAN78] pois ela resume a idéia central da abordagem proposta. No caso de substituição de versões de classes, é usada uma adaptação da tradicional técnica de blocos de recuperação. Esta adaptação dispensa o uso de um teste de aceitação e emprega apenas duas alternativas: (a)a alternativa primária é a versão antiga, representada pelo objeto O1 e a alternativa secundária é versão nova, representada pelo objeto O2. A versão O1 é considerada correta e é usada como teste de aceitação, pois seus resultados são usados para fins de observação do comportamento da nova versão. Após a execução da versão primária, a outra alternativa é executada e seus resultados comparados com os fornecidos pela versão primária. Cada vez que a nova versão O2 apresentar um funcionamento ou resultado incorreto, é fornecido para o cliente o resultado da versão antiga.

Considerando a hipótese de a substituição estar sendo realizada para corrigir alguma falha na versão antiga, O1, existe a possibilidade de ser fornecido um resultado incorreto, gerado de forma falha pela versão O1 e que divergiu do resultado fornecido pela nova versão O2 (possivelmente correto). Para detectar este tipo de ocorrência, é gravado um arquivo contendo todos os resultados gerados por ambas as versões, durante todo o processo de transição.

3 O Processo de Substituição

Um computador servidor executa a versão antiga do objeto O1 que realiza as suas funções de serviço, supostamente de forma correta e confiável. Quer se substituir dinamicamente este objeto O1 pelo objeto O2. Temporariamente, durante o processo de validação, vai de fato ser executado um programa de validação "PV" que contém simultaneamente ambas as versões O1 e O2 e que vai passar a gerenciar estes dois objetos, além de executar as tarefas de validação, como esquematizado na Figura 1.

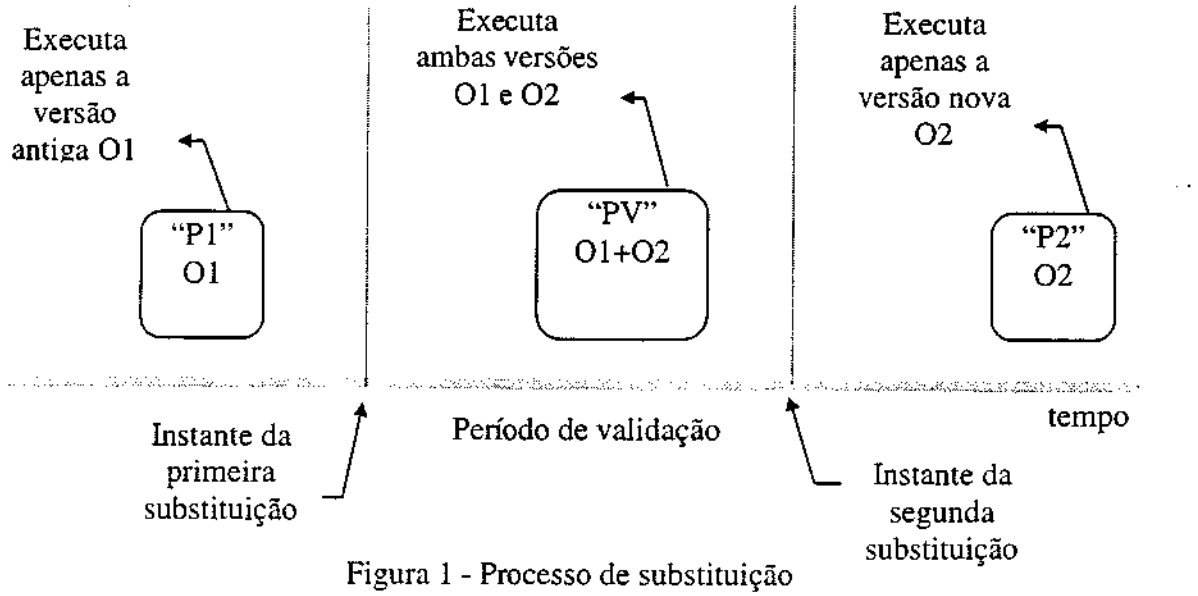


Figura 1 - Processo de substituição

3.1 Salvamento do Estado da Computação

Uma substituição no modelo de objetos apresenta possibilidade de realizar trocas no código e também nos tipos de dados, que são as propriedades definidas em cada classe. Por outro lado, um objeto possui um estado interno [BOO94] que se modifica a cada ativação. Assim, deve haver preocupação com a preservação do estado do objeto, cujos valores das variáveis instanciadas pelo objeto antigo precisam ser mapeados para os valores correspondentes no novo domínio , principalmente quando o novo objeto exigir uma reestruturação de dados, por ser oriundo de outra classe. Este estado interno deve ser salvo antes do cancelamento do objeto primário O1 e restaurado quando o objeto voltar à execução, agora instanciado dentro do programa de validação PV, que passará a gerenciar também o objeto que deverá substituir O1. O salvamento de estado de um objeto neste contexto resume-se a manter uma cópia dos dados de instância; a posterior recuperação do objeto original é feita por simples atribuição. Já a transferência de estado para outro objeto pode envolver operações de conversão, caso as estruturas de dados não sejam idênticas às do objeto que originou o salvamento de estado.

3.2 Pontos de Recuperação

Um dos serviços básicos de técnicas de TF é restaurar o sistema a um estado consistente [JAL94]. Baseada nesta afirmação está a estratégia de recuperação adotada neste modelo.

O programa PV contém meta-objetos associados a O1 e a O2 e que fazem o papel de árbitro da substituição. O meta-objeto (MO) associado a O2 toma conhecimento da falha de O2 através da comparação dos seus resultados com os resultados corretos fornecidos por O1. O meta-objeto assume as funções de TF, fazendo a detecção de falhas, o confinamento de danos, registro de erros num arquivo de log, e envia uma mensagem para O1 transferir o seu estado para que O2 consiga restaurar-se e novamente apresentar um estado correto e consistente, podendo retornar à operação normal e ser submetido a novas ativações e continuar sendo validado sob novas circunstâncias. Assim o estado incorreto de O2 é substituído pelo estado correto de O1. A Figura 2 ilustra a seqüência de operações de salvamento e restauração de estados.

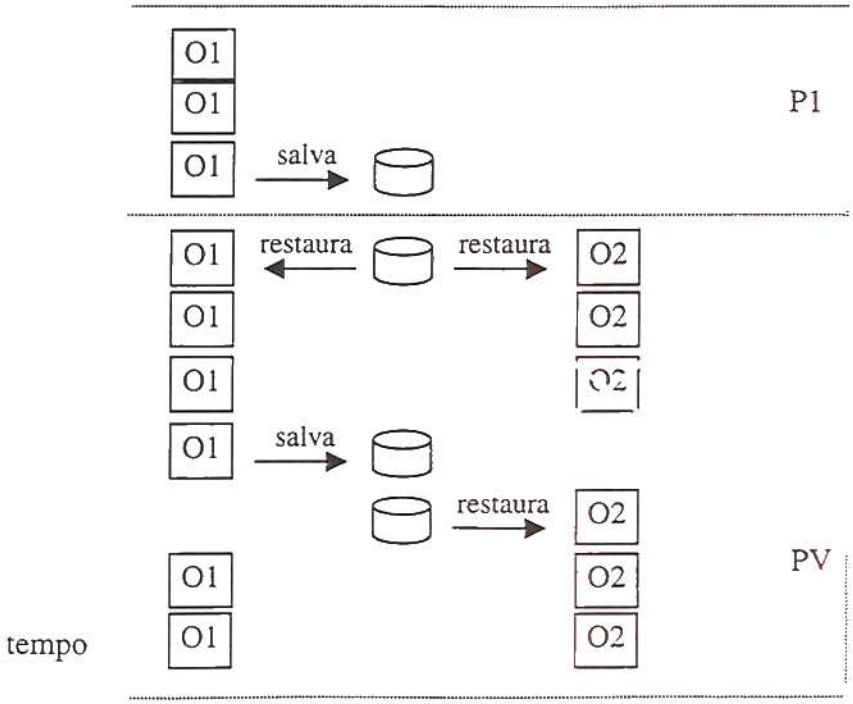


Figura 2 - Pontos de salvamento e restauração

4 Estudo de Caso

O cenário básico escolhido para desenvolver a aplicação com a qual foram feitos os testes de substituição de versão, comumente aparece na literatura e compreende um sistema de tempo-real adaptado de [BUR96] Consiste de um modelo de um sistema de drenagem para uma mina, que controla o nível de água de um reservatório através de um sensor. Uma bomba faz a coleta de água de um recipiente no fundo de um poço e a transporta para a superfície. O sistema possui as características tipicamente embutidas nos sistemas de tempo-real e pode ser implantado em uma arquitetura distribuída ou centralizada. Para este estudo de caso, optou-se pela substituição do objeto controlador do sensor. O sensor controlado pelo objeto O1 deve ser substituído por um novo sensor físico, controlado pelo sensor abstrato representado pelo objeto O2.

A linguagem de programação utilizada para a implementação foi Java e o modelo de reflexão empregada foi inspirado no protocolo de MetaJava [GOL97], visto que o protocolo de reflexão do ambiente JDK1.1® á basicamente introspectivo, não oferecendo muitas facilidades para intervenção no estado da computação. Java foi selecionada como linguagem de implementação por permitir carga de classes durante o processo de execução e possuir facilidades para programação concorrente e distribuída.

A aplicação é do tipo duas camadas ('two tiers'). O cliente e o servidor comunicam-se através de 'sockets', usando as classes Socket e ServerSocket importadas da biblioteca java.net. O 'socket' faz uma ligação virtual entre o servidor e o cliente. Ao ser efetuada uma substituição do objeto servidor antigo pelo novo, a conexão entre o cliente e o objeto servidor antigo é temporariamente desfeita e em seguida restabelecida pelo objeto que executa a nova versão. O programa cliente não é interrompido durante este processo de substituição.

Neste trabalho, para simular a ocorrência de falhas nas versões candidatas à substituição, foi usada a injeção de falhas por software, através de alterações no conteúdo de registros e variáveis. Para testar a implementação foram simuladas as seguintes situações de falha: (a) falha na leitura fornecida pelo sensor O2; (b) falha no algoritmo que realiza o cálculo da média das leituras.

Os testes realizados mostraram a viabilidade da solução proposta. O objeto servidor teve sua versão substituída sem descontinuar o cliente. As divergências entre os resultados fornecidos por O1 e O2 foram registradas no arquivo de log e o cliente sempre recebeu a resposta correta fornecida por O1, mesmo na presença de falhas em O2. Assim o objeto O2 foi testado e validado sem propagar erro para o sistema como um todo.

5 Conclusões

O objetivo deste trabalho foi de estudar e propor uma estratégia para substituição dinâmica de versões de componentes de software tolerante a falhas orientado a objetos. Este objetivo foi atingido e foi demonstrada a viabilidade de construção de um sistema com esta finalidade. O trabalho apresenta uma solução genérica para o problema da substituição sendo aplicável, em sua essência, a sistemas centralizados, distribuídos, ou tempo real, desde que o sistema seja concebido com as características propostas: adote o modelo de objetos, com componentes sujeitos a alterações estruturados na forma de classes e que o ambiente de execução ofereça a possibilidade de carga dinâmica de classes.

® Sun Microsystems

O problema apresentado é relevante visto que a manutenção de software é uma realidade e causa interrupções inconvenientes aos usuários dos sistemas. A solução proposta é inovadora no sentido de ser voltada para o modelo de objetos, utiliza técnicas de tolerância a falhas e adota uma arquitetura reflexiva. As técnicas de tolerância a falhas são usadas para validar a substituição, fazendo o monitoramento de ambas as versões através da comparação dos resultados por elas fornecidos.

O emprego da reflexão computacional permite separar o código das rotinas de substituição e validação dos demais componentes da aplicação, o que torna mais fácil compreender, manter e depurar o programa em si.

5 Referências Bibliográficas

[BOO94] BOOCH, G., "Object-Oriented Analysis and Design With Applications", The Benjamin/Cummings Publishing Co., USA, Second edition, 1994.

[BUR96] BURNS, Alan, and Wellings Andy, "Real-Time Systems and Programming Languages", Second Edition, Addison Wesley, 1996.

[FRZ97] FRANZ, Michael, "Dynamic Linking os Software Components", Computer, IEEE, pp. 74-81, March, 1997.

[GOL97] GOLM, Michael, "Design and Implementation of a Meta Architecture for Java", Institut für Mathematische Maschinen und Datenverarbeitung der Friedrich-Alexander-Universität, Erlangen-Nürnberg, Diplomarbeit, Jan, 1997.

[GUP93] GUPTA, Deepak; JALOTE, Pankaj; "Dynamic Software Version Change Using State Transfer Between Processes", Software Practice and Experience, Vol 23, pp. 949-964, September, 1993.

[GUP96] GUPTA, Deepak; JALOTE, Pankaj; and BARUA, Gautam; "A Formal Framework for Dynamic Software Version Change", IEEE Transactions on Software Engineering, Vol 22, No. 2, pp. 120-131, February, 1996.

[JAL94] JALOTE, Pankaj., "Fault-Tolerance in Distributed Systems", Prentice-Hall, New Jersey, 1994.

[LIS97] LISBÔA, M.L.; HAETINGER, W. "Troca Dinâmica de Componentes: problemas e soluções no modelo OO". Argentine Simposium on Object-Orientation, Buenos Aires, anais pp.67-75, setembro, 1997.

[RAN78] RANDELL, B.; LEE, P. A. e TRELEAVEN, P. C., "Reliability Issues in Computing System Design", ACM Computing Surveys, New York, v.10, n. 2, p. 123-166, Jun, 1978.

[SEG93] SEGAL, M., Mark E., Frieder O., "On-the-fly Program Modification: Systems for Dynamic Updating", IEEE Software, vol. 10, num. 3, pp. 53-65, March, 1993.

Experimentos de Tolerância a Falhas em Java

Maria Lúcia B. Lisbôa
Instituto de Informática
llisboa @ inf.ufrgs.br

Werner Haetinger
CPGCC
wernerh @ inf.ufrgs.br

Gustavo Canto da Silva
CBCC
canto @ inf.ufrgs.br

Universidade Federal do Rio Grande do Sul
Instituto de Informática
Av. Bento Gonçalves, 9500 - Bloco IV
CEP 91501-970 Porto Alegre - RS

Resumo

No modelo de objetos, programas são estruturados a partir de componentes encapsulados e que interagem através de interfaces bem definidas. A interação dos componentes depende fortemente da estrutura adotada no programa ou sistema, bem como seu cenário de execução: seqüencial, paralelo ou distribuído. Entre as condições propícias à manifestação de uma falha, o meio-ambiente desempenha um papel importante e, portanto, deve ter a sua atuação bem delimitada. Um meio-ambiente desfavorável pode ocasionar diferenças de comportamento em duas cópias idênticas do mesmo software. Pequenas diferenças nas máquinas virtuais onde o software é executado podem ser suficientes para a manifestação de uma falha. É precisamente nas semelhanças e diferenças de diversas formas de interação de componentes e seus distintos ambientes de execução que este trabalho concentra seus experimentos, estudando a adequação da linguagem Java para a implementação de programas tolerantes a falhas.

Abstract

In the object model, programs consist of encapsulated components, which interact through well-defined interfaces. The interactions among the components depend on the current system or program architecture as well as the execution environment: sequential, parallel or distributed. Also aspects of the environment in which the system operates cannot be ignored by the system specification. An improper environment can trigger a component/interaction fault, so its important to bound the system influence over the running program. Even when executing replicas of the same base component, small differences in the system behavior can lead the program to a faulty state. This paper is actually about the similarities and differences among the several ways the components interact and their execution environment. A major goal is to test the adequacy of Java programming language and its several virtual machines to implement fault-tolerant applications.

1 Introdução

O desenvolvimento de software tolerante a falhas no modelo de orientação a objetos se apresenta como uma estratégia promissora para a construção de software com requisitos de alta confiabilidade [BUZ97], por razões inerentes ao próprio modelo. A habilidade de classes de herdar propriedades, de forma integral ou seletiva, a possibilidade de aumentar, modificar ou anular esta herança, viabiliza a programação por diferença [GOL83]. Apenas as propriedades distintas necessitam ser implementadas, mantendo intocadas as demais propriedades. Uma conseqüência imediata da programação por diferença é a manutenção da confiabilidade dos componentes já existentes, verificados e/ou testados pelo uso.

Falha de software é qualquer imperfeição presente no código executável de um programa, cuja manifestação possa implicar a diminuição de sua confiabilidade. Mesmo presente, uma falha pode passar despercebida durante todo o processo de desenvolvimento do software e mesmo por longos períodos de sua vida útil pode permanecer latente, caso não ocorram as condições propícias à sua manifestação. Entre as condições propícias à manifestação de uma falha, o meio-ambiente desempenha um papel importante e, portanto, deve ter a sua atuação bem delimitada. Um meio-ambiente desfavorável pode ocasionar diferenças de comportamento em duas cópias idênticas do mesmo software. Por exemplo, pequenas diferenças nas máquinas virtuais onde o software é executado podem ser suficientes para a manifestação de uma falha.

É precisamente nas semelhanças e diferenças de distintos ambientes de execução que este trabalho concentra seus experimentos, estudando a adequação da linguagem Java para a implementação de programas tolerantes a falhas. Para a obtenção de componentes redundantes, uma mesma funcionalidade é implementada de três formas distintas, através de classes Java, usando diversidade de métodos em uma única classe e diversidade de classes, separadamente compiladas. Estes componentes são submetidos à execução em diferentes máquinas virtuais, com o intento de observar diferenças no meio-ambiente de execução Java. Distintas formas de execução de componentes - seqüencial e concorrente, também são objeto de estudo, para fins de observação dos tempos de execução.

2 Componentes de programas

Programas são organizados a partir de componentes individuais, que implementam as diversas funcionalidades previstas na especificação do programa ou sistema. Componentes de software oferecem a possibilidade de múltiplas execuções independentes. A informação exigida para a execução de um componente pode ser decomposta em duas partes: uma parte permanente, que consiste da estrutura dos dados e da seqüência de instruções a serem executadas, e uma parte temporária, que compreende valores de dados e outras informações contextuais que variam a cada execução do componente.

A parte permanente ou estrutural, uma vez determinada pelo projeto do componente, permanece invariante, e as correspondentes estruturas de dados e as instruções são reutilizadas ao longo das diversas execuções. Além disso, podem ser transportadas para outros programas ou sistemas sem exigir alterações. A parte temporária ou comportamental é ditada pela interação dos componentes e determina, a cada instante, o estado da execução do componente, incluindo valores de dados, de registradores e outras informações mantidas durante o processo de execução. Portanto, a parte temporária é específica de cada interação entre componentes e se constitui no diferencial mais importante entre execuções de um mesmo componente[LIS97a]

Mais especificamente, no modelo de objetos, os componentes são representados por classes de objetos, que, sob o ponto de vista estrutural, são considerados como componentes atômicos de um programa, uma vez que classes gozam da propriedade de encapsulamento. Sob o ponto de vista de execução, os objetos (instanciados a partir das classes) respondem

pelo estado da execução. Outrossim, tão importante quanto cada componente individual de um programa é o modo como estes componentes interagem para atender os serviços esperados do programa [SHA95]. A interação dos componentes depende fortemente da estrutura adotada no programa ou sistema, bem como seu cenário de execução: seqüencial, paralelo ou distribuído. A seguir são sumarizadas as principais formas de concepção, execução e interação de componentes.

2.1 Concepção de objetos

Um objeto é uma abstração descrita em uma classe e delineada a partir de seus requisitos funcionais - o que ele deve fazer - e não funcionais - o que ele deve observar além de sua funcionalidade específica. Entre os requisitos não funcionais de um sistema incluem-se: adaptabilidade, interoperabilidade, eficiência, testabilidade, reutilização e confiabilidade. De acordo com Bushmann et al. [BUS96], a confiabilidade inclui os aspectos de robustez e tolerância a falhas.

Uma abstração de um objeto começa a ser esboçada a partir de dados, serviços, interfaces e, antecipando a sua utilização, o ambiente de execução. O projeto de tolerância a falhas deve ser feito neste momento, principalmente quando envolver projeto diversitário. A tolerância a falhas em software usando redundância de componentes exige nesta fase a implementação de métodos ou objetos diversificados, derivados ou não da mesma classe ancestral, e a implementação de técnicas de gerenciamento de redundância.

Técnicas de gerenciamento de redundância tem por objetivo controlar a execução de cada serviço solicitado a objetos replicados ou diversificados, fornecendo uma única resposta, supostamente confiável, ao cliente desse serviço. A forma de interação dos objetos envolvidos e o ambiente de execução desempenham um papel muito importante na questão de gerenciamento da redundância.

2.2 Cenários de execução: seqüencial e concorrente

Na execução seqüencial e centralizada, um componente requisita a execução de outro componente através do envio de uma mensagem, interrompendo a sua execução enquanto aguarda o término da execução do componente requisitado. A figura 1 esquematiza o fluxo de execução por requisição e serve como cenário básico para introduzir a idéia de componentes com propriedades de tolerância a falhas.

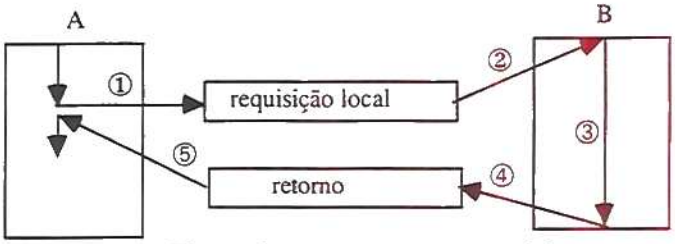


Figura 1: Fluxo de execução por requisição

Neste cenário destacam-se os seguintes pontos demarcados na figura 1 e que indicam momentos de execução e suas possibilidades de falha.

(1) O componente A requisita a execução do componente B, por meio de uma mensagem parametrizada. Falha na mensagem de requisição. Ex. Nome incorreto do método.

(2) O componente B recebe a requisição, identifica o método destinatário da mensagem e inicia a sua execução. Falha nos parâmetros de requisição. Ex. Tipo incorreto de parâmetro.

(3) O componente B executa as operações contidas no método selecionado. Falha em operação interna de B. Ex. divisão por zero.

(4) O componente B termina a sua execução e retorna resultados. Execução com erro. Ex. Retorna indicação de erro.
(5) O componente A recebe os resultados da execução de B e retoma a sua execução no ponto de interrupção. Resultados incorretos da execução de B. Ex. Valor fora dos limites.

A linguagem de implementação pode contribuir para prevenir a ocorrência das falhas mencionadas nos momentos (1) a (3), caso possua mecanismos como verificação estática e/ou dinâmica de tipos e mecanismos de tratamento de exceções. No caso de Java, estas características estão presentes. Já os tipos de falhas descritas em (4) e (5) devem ser abordados através de técnicas de programação tolerante a falhas, visto que o componente requisitado foi incapaz de fornecer o serviço solicitado. A solução consiste em solicitar o mesmo serviço a outro componente, pressupondo a existência de redundância.

A execução concorrente de componentes estende o cenário seqüencial para vários fluxos de execução: vários componentes de um mesmo programa podem ser simultaneamente executados (ou dar esta impressão ao cliente). Ou ainda, o mesmo componente pode ser submetido a mais de um fluxo de execução, atendendo a diferentes requisições. Em Java, a programação concorrente pode ser implementada através de *threads* de execução. Em alguns ambientes de execução Java, esta forma de implementação possui atualmente algumas importantes limitações, tais como: (a) dificuldade de obtenção de paralelismo real, visto que todos os *threads* são executados em um mesmo processador; (b) ao final da execução de um *thread*, o objeto é destruído, necessitando ser criada uma nova instância para posterior execução[LIN96].

3 Estudo de caso

Tendo por objetivo principal observar diferenças na forma de interação dos componentes e no meio-ambiente de execução Java, uma mesma funcionalidade é implementada de três formas distintas, por diversidade de métodos e diversidade de classes. A partir destas classes, foram estruturados três programas, com distintas arquiteturas: o primeiro utiliza uma única classe, com diversidade de métodos, o segundo reúne as três classes, separadamente compiladas e o terceiro utiliza as mesmas classes, com *threads* de execução. Estes programas foram submetidos à compilação e execução em diferentes máquinas virtuais, com o intento de medir os respectivos tempos de execução.

Para a implementação da redundância foram selecionados três métodos numéricos de interpolação polinomial. Esta escolha deve-se à facilidade de implementação dos algoritmos de interpolação e a grande probabilidade de divergência de resultados, por problemas inerentes aos próprios métodos bem como erros gerados no processo de cálculo. Tais características são úteis para testes de algoritmos de votação e de aceitação de resultados, usados pelas técnicas de programação em n-versões e blocos de recuperação.

Experimento 1: Execução seqüencial

a) O primeiro programa, executado como um 'applet', utiliza diversidade de métodos: cada algoritmo de interpolação é implementado como um dos métodos de uma mesma classe. Os três métodos usam idênticos argumentos:

```
r1= this.lagrange(vet,10,aux.doubleValue());
r2= this.linear(vet,10,aux.doubleValue());
r3= this.newton(vet,10,aux.doubleValue());
```

b) No segundo programa, os objetos diversificados são implementados pelas classes Linear, Lagrange e Newton. A classe principal instancia e executa os objetos diversificados:

```
double r1= newt.method(vet,n,aux.doubleValue());
double r2= lag.method(vet,n,aux.doubleValue());
double r3= lin.method(vet,n,aux.doubleValue());}
```

Experimento 2: Execução concorrente

Para este exemplo, o programa (c) foi estruturado com as mesmas três classes do programa (b), porém os objetos são executados através de *threads*. A classe principal continua responsável pela instanciação dos objetos, execução dos métodos e coleta dos resultados.

```
newt= new newton(this,aux.doubleValue(),vet,n);
lag= new lagrange(this,aux.doubleValue(),vet,n);
lin= new linear(this,aux.doubleValue(),vet,n);
lag.start();
newt.start();
lin.start();
```

3.1 Resultados dos experimentos

Inicialmente, os programas (a) e (b) foram submetidos à execução seqüencial em diferentes ambientes, usando máquinas semelhantes em configuração e foram coletados os tempos de tempos de execução. Os experimentos para medição de tempo foram realizados com 50 execuções de cada método ou objeto, para potencializar eventuais divergências, bem como minimizar o efeito do tempo de carga do programa, observado na primeira execução.

O efeito do tempo de carga não pode ser desprezado, como pode ser visto na Tabela 1, que registra o tempo de uma execução e o tempo médio de 50 execuções do mesmo método ou objeto. Os resultados registrados na tabela 1 forma obtidos através de uma média de 10 ativações, para minimizar a influência do momento de execução.

Tabela 1 - Tempo médio de execução

Ambiente	Média - 1 execução	Média - 50 execuções
PC/JDK 1.1.	331 ms	13 ms
PC/ Visual J++	204 ms	3.4 ms
PC/Borland Suite/Java	210 ms	10 ms
PC/ Visual Café	29 ms	4.4 ms

A seguir, foram repetidos os testes com execução paralela, utilizando *threads*. Em Java*, a execução concorrente é obtida através de herança da classe *Thread*. Um *thread* de execução deve ser explicitamente ativado para um determinado objeto e, quando encerra a execução, normal ou excepcional, o objeto é destruído. Portanto, para a obtenção de diversos *threads* de execução, deve ser criado um novo objeto e feita uma nova ativação. Os resultados obtidos encontram-se esquematizados na tabela 2.

Tabela 2 - Execução seqüencial e concorrente

Ambiente	Programa (a) - execução seqüencial	Programa (b) execução seqüencial	Programa (c) Execução concorrente
PC/JDK 1.1.	3.7 ms	3.2 ms	32 ms
PC/ Visual J++	3.4 ms	2.2 ms	3.4 ms
PC/Borland Suite/Java	10 ms	5.5 ms	21 ms
PC/ Visual Café	4.4 ms	2.2 ms	4.4 ms

A linguagem Java se caracteriza pela universalidade do código gerado pelos diferentes tradutores e que pode ser executado em qualquer ambiente, sem necessidade de alteração e mesmo de recompilação. Durante os testes o mesmo código fonte foi usado, porém algumas vezes foi recompilado nos diferentes ambientes, para observar se o tamanho do código gerado apresentava diferenças. Constatou-se que o tamanho do código gerado nos diversos ambientes

* Ambiente JDK1.1

não apresentava diferenças significativas, principalmente devido ao pequeno número de classes que compõem os programas do experimento.

Quanto à forma de execução, seqüencial ou concorrente, os experimentos para medição de tempo mostraram divergências significativas nos diferentes ambientes. Os tempos obtidos mostraram que alguns ambientes praticamente não apresentaram diferenças nos tempos de execução seqüencial e concorrente, a exemplo de J++ e Visual Café, enquanto outros apresentaram o tempo de execução concorrente superior ao tempo de execução seqüencial. Hipóteses para justificar estas diferenças são: (a) o tempo de criação e destruição dos objetos submetidos a *threads* de execução, e, (b) tipo de suporte para a execução dos *threads*, oferecido pela máquina virtual ou sistema.

4 Conclusões e trabalhos futuros

Os experimentos sobre a adequação da linguagem Java para fins de tolerância a falhas têm mostrado resultados encorajadores em relação à portabilidade da linguagem e à existência de mecanismos que possibilitam a implementação segura e transparente de requisitos não-funcionais. Em trabalhos anteriores [LIS97b] foram realizados testes com carga dinâmica de classes e com reflexão computacional [HAE98].

Neste trabalho, os experimentos de medição de tempos de execução foram feitos com informações obtidas a partir do relógio do sistema, cuja precisão pode não ser suficiente para capturar diferenças de tempo em programas pequenos, com os do presente estudo de caso.

Na continuidade, serão feitos experimentos com programas maiores e com maior número de interações e execuções e novos experimentos envolvendo componentes:

(a) com execução distribuída usando sockets e RMI - Remote Method Invocation. O objetivo dos testes, além da medição de tempos de execução, é a avaliação de chamadas remotas parametrizadas, com diferentes tipos de dados.

(b) com componentes auto-protegidos. O mecanismo mais importante de auto-proteção é o mecanismo de *tratamento de exceções*. Através dele, a possibilidade de ocorrência de vários tipos de falhas pode ser antecipada, erros podem ser detectados durante o processo de execução e providências locais (internas ao objeto) podem ser tomadas com vistas à continuidade do serviço, ou mesmo, quando não é possível resolver internamente, o objeto pode sinalizar ao cliente de seus serviços a ocorrência de uma situação excepcional que o impede de fornecer o serviço solicitado.

Bibliografia

[BUS96] BUSCHMANN, F. et al. A System of Patterns: pattern-oriented software architecture. John Wiley & Sons, England, 1996.

[BUZ97] BUZATO, L. E.; RUBIRA, C.M.F.; LISBOA, M. L. A reflective Object-oriented architecture for developing fault-tolerant software. Jornal Of the Brazilian Computer Society, Vol. 4, No. 2; November 1997, p. 39-48.

[GOL83] GOLDBERG, A. The influence of an object-oriented language on the programming environments. In: ACM COMPUTER SCIENCE CONFERENCE, 1983, Orlando, Florida, USA. **Proceedings**...New York: ACM, 1983. p. 35-54.

[HAE98] HAETINGER, W.; LISBÔA, M.L. Substituição dinâmica de classes com validação de objetos. Trabalho submetido ao I Workshop de Tolerância a Falhas, Porto Alegre, maio de 1998.

[LIN96] LINDEN. P. v. Just Java. Sunsoft Press, Mountain View, CA, USA, 1996.

[LIS97a] LISBOA, M. L. Arquiteturas de meta-nível. Tutorial. Simpósio Brasileiro de Engenharia de Software, Fortaleza, CE, 1997.

[LIS97b] LISBÔA, M.L.; HAETINGER, W. Troca Dinâmica de Componentes: problemas e soluções no modelo OO. Argentine Simposium on Object-Orientation, Buenos Aires, anais pp.67-75, setembro, 1997.

[SHA95] SHAW, M; GARLAN, D. Formulations and Formalisms in Software Architecture. Lecture Notes in Computer Science, n. 1000, Berlin: Springer, 1995.