Experiência com a Implementação de um Injetor de Falhas em Linux

Fábio Olivé Leite[1]
(olive@conectiva.com.br)
Conectiva S/A
R. Tocantins, 89
Curitiba - PR

Taisy Silva Weber
(taisy@inf.ufrgs.br)
PPGC - UFRGS
Av. Bento Gonçalves, 9500
Porto Alegre - RS

Resumo

Este artigo trata da validação experimental de mecanismos de Tolerância a Falhas através da Injeção de Falhas, relatando a experiência obtida durante a implementação da ferramenta ComFIRM. É feita uma pequena revisão dos conceitos genéricos de Injeção de Falhas (por simulação, hardware e software) e então são dissecadas as técnicas de implementação de injetores de falhas em software, considerando suas vantagens e desvantagens. É dada especial atenção à Injeção de Falhas no nível do Sistema Operacional.

Palavras-chave: Injeção de Falhas, validação experimental, Tolerância a Falhas.

Abstract

This paper is about the experimental validation of Fault Tolerance mechanisms through Fault Injection, reporting the expertise gathered during the implementation of ComFIRM. A small revision of the generic concepts behind Fault Injection (via simulation, hardware and software) is given, and then the software fault injectors implementation techniques are approached, considering their advantages and disadvantages. Special attention is given to Operating System level Fault Injection.

Keywords: Fault Injection, experimental validation, Fault Tolerance.

1 Introdução

Vivemos em uma sociedade cada vez mais dependente de recursos computacionais, principalmente de Sistemas Distribuídos, mesmo para alguns de seus serviços mais vitais. Nenhum destes sistemas está totalmente livre da ocorrência de falhas, portanto é necessário que contenham mecanismos de Tolerância a Falhas para garantir seu funcionamento correto, ou pelo menos a degradação não catastrófica de seus serviços.

Mesmo estes mecanismos de Tolerância a Falhas devem ser testados e depurados, pois de nada adianta um algoritmo que mascare ou recupere um determinado erro e crie outros. Algoritmos e protocolos podem ser testados e comprovados formalmente através de técnicas de matemática e lógica, provadores de teoremas e assim por diante. Assim se pode garantir a correção das

[1]Fábio Olivé Leite é mestrando do Programa de Pós-Graduação em Computação da UFRGS, sob orientação da Profª Drª Taisy Silva Weber, na área de Tolerância a Falhas.

especificações, porém um passo adicional ainda é necessário para estas especificações serem efetivamente utilizadas, a implementação.

A história nos mostra que mesmo bons programadores cometem erros, portanto nada garante que de uma *especificação* correta sempre se obtém uma *implementação* correta. Assim como se pode testar e comprovar formalmente uma especificação se pode fazer com uma implementação, porém com a atual complexidade dos sistemas microprocessados — ainda mais dos distribuídos — é realmente impossível se antecipar e verificar, em tempo razoável, toda a miríade de possibilidades de comportamento desta implementação, quando da ocorrência de falhas.

Neste caso, a comprovação de funcionamento correto pode se dar através de uma técnica experimental, a Injeção de Falhas [ARL 90]. A Injeção de Falhas visa fornecer exatamente a entrada que os mecanismos de Tolerância a Falhas esperam, as falhas. Mais especificamente, ferramentas de Injeção de Falhas permitem que se crie um ambiente onde não só é certo que determinadas falhas ocorrerão, mas também que ocorrerão de forma controlada e que seu impacto no sistema alvo poderá ser medido e estudado posteriormente.

Este artigo contempla a implementação de injetores de falhas em software, discutindo aspectos importantes como a localização, modo de ativação, impacto do injetor no sistema e sua eficácia em simular falhas reais. A experiência aqui relatada é decorrente da implementação da ferramenta ComFIRM [BAR 99], uma ferramenta de injeção de falhas de comunicação inspirada na ferramenta ORCHESTRA [DAW 96].

2 Injeção de Falhas

A Injeção de Falhas pode se dar por simulação, por hardware ou por software [HSU 97]. A injeção por simulação utiliza um modelo do sistema sendo testado, que é executado em um simulador, e portanto permite total controle sobre o que está acontecendo, bem como total possibilidade de análise do experimento. Embora pareça bastante atrativa, a injeção por simulação esbarra em dois grandes problemas.

O teste é sobre um *modelo* do sistema, portanto todas as conclusões obtidas serão sobre este modelo, não sobre o sistema em si; quanto mais próximo o modelo estiver do sistema, mais próximas da realidade estarão as conclusões. Em segundo lugar, conseguir modelos fiéis, completos, de sistemas microprocessados atuais é praticamente impossível, muitos segredos industriais estão envolvidos em qualquer processador atual.

A Injeção de Falhas por hardware pode se dar tanto com contato quanto sem contato. A injeção sem contato se utiliza de interferência eletromagnética, de radiação de íons pesados ou técnicas similares, de forma a perturbar o sistema em teste, causando falhas. Já a injeção com contato utiliza pontas de prova, sondas e soquetes para alterar níveis de tensão dentro dos circuitos, sendo a abordagem mais intrusiva [CAR 98].

Este tipo de Injeção de Falhas é o mais próximo que se pode chegar das falhas reais, visto que ocorrem no universo físico [PRA 96]. Ainda assim, existem três grandes desvantagens que levaram esta abordagem a ser deixada de lado ultimamente. Em primeiro lugar, a injeção por hardware pode causar a queima de componentes e equipamentos, de forma que os custos podem ser altos e a colheita de resultados nem sempre é possível.

A segunda desvantagem é a dificuldade de controle sobre a injeção por hardware sem contato. Não se pode saber em que ponto dos circuitos os campos magnéticos estão realmente atuando, nem qual está sendo seu efeito. Finalmente, o desenvolvimento de injetores de falhas por hard-

ware com contato é específica para o sistema em teste, e demanda muito tempo e conhecimento para ser corretamente implementada. Os injetores criados para uma plataforma dificilmente serão aproveitados para outro experimento, e novamente os custos podem ser altos.

Finalmente, a Injeção de Falhas por software é a emulação de falhas reais (do universo físico) dentro do universo informacional. Esta abordagem poderia ser chamada Injeção de Erros, visto que é só o que pode fazer, porém não deve ser subestimada. Na Tolerância a Falhas implementada em software, que é muito comum hoje em dia, a Injeção de Falhas por software é exatamente o que se necessita, visto que na verdade o software só percebe as falhas na forma de erros.

A Injeção de Falhas por software consiste da criação de código que simule a ocorrência das falhas necessárias à validação do sistema. Este código normalmente deve ser inserido em algum local onde venha a ser ativado pelo sistema em teste, para que possa injetar as falhas. Sendo assim, pode-se pensar em alguns níveis onde o injetor pode estar. Por exemplo, o injetor pode estar dentro do Sistema Operacional, o que faz com que os processos em execução não percebam a presença do injetor. O injetor pode ainda estar no mesmo nível que as aplicações, executando como um processo ou uma thread. Finalmente, pode-se pensar em colocar o injetor no meta-nível, utilizando-se ténicas da Reflexão Computacional [BAR 99].

Estes três níveis criam uma boa gama de possibilidades para a implementação de um injetor, mas as vantagens são ainda maiores. O tempo de criação de um injetor por software é bem menor que o tempo gasto em simulação ou em criação de hardware específico. O código gerado pode ser generalizado de forma a permitir sua utilização em diferentes plataformas, diminuindo os custos totais de criação e adaptação de um injetor. Experimentos com um injetor por software podem ser mais facilmente conduzidos e adaptados, e podem ter ativação mais precisa e mais flexível.

Tantas vantagens acabam por sobrepor as duas desvantagens desta abordagem: o fato da injeção ser de erros, não de falhas, e a alteração no ambiente de execução. A primeira desvantagem é inerente ao fato do injetor ser um programa, ser um código. Não há como, por exemplo, se testar um mecanismo de detecção e correção de erros de memória (ECC) apenas escrevendo nela. O mecanismo está em hardware, e é transparente ao software. Se for levado em consideração apenas a validação de mecanismos de Tolerância a Falhas em software, esta abordagem ainda assim se aplica muito bem.

A segunda desvantagem (dependendo do ponto de vista, a *única* desvantagem) também é inerente ao fato do injetor ser um código a ser executado. Este código deve estar em algum lugar e eventualmente ser executado, de forma que o sistema se comportará de forma ligeiramente diferente de quando o injetor não estiver sendo utilizado. Algumas chamadas de sistema podem demorar mais para retornar, o Sistema Operacional pode demorar alguns milésimos ou milionésimos de segundo a mais para escalonar processos, menos memória livre estará disponível.

Este impacto no sistema em teste deve ser bem controlado, e na verdade nota-se que não necessariamente é nocivo. A próxima seção trata exatamente das abordagens de implementação de injetores de falhas em software e seus impactos. Considerações são tecidas de acordo com alguns exemplos de modelos de falha a injetar. Cabe lembrar que a experiência aqui relatada decorre da implementação de uma ferramenta de Injeção de Falhas de Comunicação situada dentro do núcleo do Sistema Operacional Linux.

3 Injeção de Falhas por Software

Como já foi dito, a Injeção de Falhas por software pode se dar pelo menos em três níveis. A injeção feita no meta-nível consiste da aplicação de técnicas de Reflexão Computacional, criando acima do nível funcional (onde estão as funções, métodos e procedimentos) um nível meta que pode interceptar a computação feita no nível funcional. Desta forma, chamadas a funções podem ser interceptadas, e então parâmetros de entrada ou saída podem ser monitorados e modificados. Um perfil da execução no nível funcional pode ser obtido, e falhas podem ser injetadas com uma ativação bastante flexível.

Outra abordagem é a injeção feita no nível da aplicação. Esta consiste da criação de um processo injetor (ou thread injetora dentro de um processo em teste), que vai de alguma forma alterar o ambiente de execução do processo simulando a ocorrência de falhas. Uma possibilidade é a injeção de falhas via chamadas de sistema de depuração [GON 00]. Esta técnica é especialmente interessante por não necessitar da disponibilidade de código fonte do sistema em teste. Esta característica é vital para organizações independentes de teste e validação, que recebem sistemas prontos, fechados, cujo código fonte normalmente é coberto por patentes e registros comerciais.

Ainda no nível de aplicação, existe a possibilidade de se inserir em um programa (no código fonte) um código adicional que crie um thread extra que executará o injetor. Este é o ponto máximo de intrusão que se pode chegar, visto que altera o código fonte, altera o ambiente de execução, compartilha espaços de memória e rouba ciclos de CPU do sistema em teste. Problemas inesperados com o injetor (falhas?) podem atrapalhar todo o sistema, e pode ser necessário mudar a característica de implementação dos algoritmos, do sistema em teste, para acomodar a inspeção e possível alteração pelo injetor.

Uma abordagem de implementação interessante é a utilização de bibliotecas dinâmicas modificadas, cujo uso pode ser ativado ou desativado facilmente. Nos sistemas padrão Unix, por exemplo, a variável `LD_LIBRARY_PATH` pode conter um diretório onde se encontram bibliotecas a serem carregadas antes de se verificar os diretórios padronizados do Sistema Operacional. Isto permite que se execute um mesmo programa, sem alteração, porém que sofrerá a ação de uma biblioteca dinâmica com funções modificadas, que podem muito bem injetar qualquer tipo de erro no processo ao qual estão ligadas.

A melhor forma de implementação de um injetor de falhas em software parece ser dentro do Sistema Operacional. Isto porque um processo só conhece o mundo externo através das chamadas de sistema que o Sistema Operacional lhe proporciona. Se um processo implementa um protocolo de comunicação confiável, ele pede através de chamadas de sistema que o Sistema Operacional entregue estas mensagens. Para este processo, uma falha real na rede e uma falha injetada pelo Sistema Operacional são indistinguíveis [LEI 00]. Assim também acontece para falhas em memória e em registradores, por exemplo.

4 Injeção de Falhas em Sistemas Operacionais

A implementação de um injetor de falhas em um Sistema Operacional está obviamente ligada à arquitetura deste Sistema Operacional. Um sistema do tipo microkernel, por exemplo, mantém, através da abstração criada pela troca de mensagens, uma fronteira muito bem definida entre cada subsistema. Isto permite que se troque apenas para um processo em teste os servidores de determinados dispositivos, para que os tais lhe pareçam falhos. O Sistema Operacional GNU/Hurd, por exemplo, permite que certos processos tenham suas chamadas de sistema

substituídas apenas pela troca de servidores em tempo de execução [STA 00].

Já um núcleo monolítico, se não for bem organizado, pode ser muito difícil de ser alterado. Este não é o caso do Linux, que na verdade se apresenta bastante amigável a este tipo de alteração. Isto advém do fato de o Linux apresentar um projeto orientado a objetos, ainda que seja implementado em linguagem C. Por exemplo, a implementação do subsistema de rede é extremamente hierárquico e bem organizado, o que permitiu a rápida implantação da ferramenta ComFIRM.

Através da alteração de apenas duas rotinas foi possível inserir o injetor de falhas de comunicação, de forma a interceptar todos os pacotes de rede transmitidos e recebidos pela máquina. Este ponto está logo acima dos gerenciadores dos dispositivos de rede, portanto a falha injetada é bastante próxima de uma falha real não só do ponto de vista dos processos, mas também dos próprios protocolos de comunicação implementados em níveis superiores.

Aqui é importante salientar que a forma com que a falha é injetada pode acabar por dar um significado diferente à falha. Suponhamos uma mensagem gerada por um processo e que deve ser descartada pelo Sistema Operacional, para simular uma falha de perda de pacote na rede. Caso a falha real acontecesse, a mensagem passaria por todos os protocolos de rede utilizados, atualizando números de série e flags, por exemplo, e então seria enviada pela placa de rede, que colocaria em suas estatísticas o número de mensagens enviadas, de bytes, de pacotes com erro, etc. A mensagem seria perdida na rede, porém já teria influenciado todo o código por onde passou.

Sendo assim, se fosse escolhido descartar a mensagem no ponto mais próximo do processo, digamos na chamada de sistema `socket_call`, esta não influenciaria todo o código do protocolo TCP, por exemplo, portanto os números de série estariam errados para a proóxima mensagem. Do ponto de vista do protocolo, não houve mensagem. Isto não serve como uma emulação de falha de rede, pois a próxima mensagem será entregue normalmente e tudo proseguirá como se aquela mensagem descartada nunca tivesse sido gerada. Isto é uma falha da aplicação, não da rede.

Exatamente por esta razão para a ferramenta ComFIRM foi escolhido trabalhar no nível mais baixo possível, mas que ainda permitisse escapar dos detalhes de cada gerenciador de dispositivo. Alterar cada gerenciador, para as centenas de placas de rede suportadas, tornaria muito difícil a depuração da ferramenta, além de ser muito mais intrusivo. As rotinas alteradas são o ponto de união entre todos os gerenciadores de dispositivos de comunicação (abaixo) e todos os protocolos de transporte (acima).

O mesmo pode ser observado em outros pontos. Por exemplo, o *buffer cache* centraliza todas as operações em dispositivos de bloco (como discos rígidos e disquetes), portanto é um ponto bastante interessante para injeção de falhas em dispositivos de armazenamento. Novamente, seria incorreto injetar falhas no nível mais próximo do processo, nas chamadas de sistema do VFS — Virtual Filesystem Switch. Se isto fosse feito, os sistemas de arquivos jamais saberiam das operações, e as estruturas de dados que descrevem os arquivos não seriam atualizadas.

O que se pode notar como regra geral é que deve-se estar o mais próximo possível do hardware, porém não tão perto que o trabalho de programar e inserir o injetor tenha que ser refeito para cada gerenciador de dispositivo. Um Sistema Operacional bem projetado permite isto, como é o caso do Linux.

5 Conclusão

Espera-se com este artigo ter mostrado aspectos interessantes da implementação de injetores de falhas em software. A Injeção de Falhas por simulação tem o grave problema de não atuar no sistema em si, mas sim em um modelo. O mesmo não acontece com a injeção por hardware ou software, onde se pode trabalhar com o sistema real com variados níveis de intrusão. A Injeção por hardware tem suas desvantagens, a principal sendo o custo de desenvolvimento e operação do injetor, que pode até mesmo queimar o sistema em teste.

Já a Injeção de Falhas por software se mostra uma alternativa bastante interessante, pois usa o sistema real nos testes e tem um custo de implementação, adaptação e experimentação bastante baixo. Um injetor em software é uma ferramenta flexível, cuja ativação pode ser bem controlada. Quando feito no nível do Sistema Operacional, um injetor de falhas se torna um sistema de testes poderoso, pois pode-se chegar muito próximo das falhas reais, embora mantendo uma certa facilidade de configuração e condução de experimentos.

Referências

[ARL 90] ARLAT, J. et. al. **Fault-injection for dependability validation: a methodology and some applications**. IEEE Transactions on Software Engineering, Special Issue on Experimental Computer Science. New York, v. 16, n. 2, Fevereiro, 1990.

[BAR 99] BARCELOS, P. P; LEITE, F. O; WEBER, T. S. **Implementação de um Injetor de Falhas de Comunicação**. Anais do VIII Simpósio de Computação Tolerante a Falhas, Campinas, 1999.

[CAR 98] CARREIRA, J; MADEIRA, H; SILVA, J. G. **Xception: A Technique for the Experimental Evaluation of Dependability in Modern Computers**. Transactions on Software Engineering, v. 24, n. 2, Fevereiro, 1998.

[DAW 96] DAWSON, Scott; JAHANIAN, Farnam; MITTON, Todd. **ORCHESTRA: A Fault Injection Environment for Distributed Systems**. University of Michigan, Department of Electrical Engineering and Computer Science, Techical Report n. 318, 1996.

[GON 00] GONÇALVES, Luis C. R.; WEBER, Taisy S. **Injeção de Falhas via Depuradores**. Anais do 1° Fórum Internacional Software Livre 2000 - Workshop sobre Software Livre. Porto Alegre, Maio, 2000.

[HSU 97] HSUEH, Mei-Chen; TSAI, Timothy K.; IYER, Ravishankar K. **Fault Injection Techniques and Tools**. Computer, v. 30, n. 4, Abril, 1997.

[LEI 00] LEITE, Fábio O.; WEBER, Taisy S. **Injeção de Falhas de Comunicação em Linux**. Anais do 1° Fórum Internacional Software Livre 2000 - Workshop sobre Software Livre. Porto Alegre, Maio, 2000.

[PRA 96] PRADHAN, Dhiraj. **Fault-Tolerant Computer System Design**. Prentice-Hall, 1996.

[STA 00] STALLMAN, Richard. Palestra proferida no 1° Fórum Gnu/Linux de Curitiba, 30 de abril de 2000.