FIJI – Uma Ferramenta para Validação Experimental do XFS

Leonardo Garcia de Mello, Taisy Silva Weber

Instituto de Informática – Universidade Federal do Rio Grande do Sul (UFRGS)
Caixa Postal 15.064 – CEP 91.501-970 – Porto Alegre – RS – Brazil
lmello@inf.ufrgs.br

Resumo. *Sistemas de arquivos são componentes essenciais para servidores de alta disponibilidade, sendo preferível o uso daqueles que sejam seguros e relativamente independentes de ações por agentes humanos para a recuperação. Uma das abordagens para alcançar alta disponibilidade em sistemas de arquivos é a do tipo journaling, ou log de metadados. ReiserFS, ext3, JFS, e XFS são exemplos de implementações de journaling para Linux. Este artigo apresenta um injetor de falhas baseado em recursos de depuração para validação experimental do mecanismo de journaling. São mostradas as técnicas para construção do injetor e um teste aplicado sobre o XFS como sistema alvo.*

1. Introdução

O termo comercialmente conhecido como "alta disponibilidade" (*HA* de *High Availability*) representa uma característica de sistemas computacionais projetados para evitar ao máximo as interrupções, planejadas ou não, na prestação de serviços. Em alta disponibilidade, o ideal é haver poucas falhas e, mesmo quando estas acontecerem, que o seu tempo médio de reparo (ou *MTTR*, de *Mean Time To Repair*) seja tão pequeno quanto possível.

Atualmente Linux é muito utilizado em servidores de redes [BAR 2000], em um cenário onde demandas quanto à disponibilidade geralmente são bastante elevados. Praticamente todas as plataformas atuais dispõem de sistemas de arquivos com alta disponibilidade [SEL 2000]. Para uso em aplicações de missão crítica são usados sistemas de arquivos baseados em *journaling* tais como ext3, JFS, ReiserFS e XFS para o Linux.

Este artigo propõe uma estratégia de validação experimental para avaliar a eficiência do mecanismo de *journaling* para sistemas de arquivos através de injeção de falhas por *software* [HSU 97]. Esta estratégia visa como sistema alvo uma implementação do XFS, um sistema de arquivos baseados em *journaling* para o Linux e é apoiada por uma ferramenta para injeção de falhas para sistemas de arquivos baseados em *journaling* – o FIJI.

2. Integridade para dados e metadados

Informações são organizadas em disco na forma de dados e metadados. Durante as operações envolvendo dados e metadados, é preciso que a representação do sistema de arquivos em disco seja mantida consistente – isso mesmo após a ocorrência de falhas. Em sistemas de arquivos como o ext2, construídos com alocação baseada em blocos, o sistema de arquivos pode ficar inconsistente após a ocorrência de falhas porque dados e metadados são gravados em disco de forma assíncrona. Sempre que inconsistências acontecem, é preciso executar algum utilitário de verificação - como o fsck [BAR 2000].

O fsck realiza uma série de verificações ao longo de todo o sistema de arquivos para validar as suas entradas e assegurar-se de que blocos alocados em disco estão todos sendo referenciados corretamente. Mas, para discos de grande volume, a execução do fsck pode consumir um tempo demasiadamente elevado comprometendo a disponibilidade do servidor. Em sistemas com muitos *gigabytes* em arquivos, por exemplo, a execução do fsck pode consumir até 10 horas ou mais [BAR 2000]. Conforme a severidade do caso, pode ser necessária a presença física do administrador do sistema para informar a senha de *root* e executar manualmente o programa de verificação.

Sistemas de arquivos para alta disponibilidade surgiram para contornar essas dificuldades. Por meio deles, é possível diminuir a chance de serem introduzidas inconsistências entre dados e metadados e reduzir o tempo médio de reparo (*MTTR*) na ocorrência de falhas. Atualmente as principais técnicas empregadas para auxiliar a obter alta disponibilidade em sistemas de arquivos Unix são o *journaling* e o *Soft Updates* [SEL 2000].

Enquanto a técnica de *Soft Updates* apenas é utilizada em sistemas BSD, a técnica de *journaling* baseia-se na redundância para aumentar a confiabilidade dos dados e metadados - mas sem aumentar significativamente os custos de hardware. Ela já é adotada por sistemas de arquivos em sistemas operacionais para plataformas diversas, tais como Solaris, AIX, Digital UNIX, HP-Ux, Irix e Windows NT [SEL 2000]. Mas apesar de os sistemas de arquivos baseados em *journaling* terem sido adotados quase como um padrão pela indústria de *software*, atualmente não se encontram publicações sobre avaliação de medidas da sua disponibilidade.

3. Sistemas de arquivos baseados em journaling

Sistemas de arquivos baseados em *journaling* fazem um controle sobre as mudanças realizadas nos metadados (JFS, ReiserFS e XFS), ou nos dados e metadados (ext3) associados a um sistema de arquivos. A idéia consiste em tratar diferentemente dados e metadados, usando uma área dedicada em disco (o *log*, ou *journal*) para manter um histórico das mudanças. Tais sistemas implementam uma política de *write-ahead logging*, fazendo com que registros sejam armazenados no *log* antes de as operações serem efetuadas [SEL 2000].

Um sistema de arquivos deste tipo pode ser empregado para aplicações que lidem principalmente com um grande número de arquivos pequenos e façam uso freqüente da chamada de sistema *sync*. Isso acontece porque as várias alterações nos metadados são transferidas para o disco através de uma única operação, que acrescenta várias transações em uma mesma área [SEL 2000].

Uma grande vantagem da abordagem baseada em *journaling* é de que ela facilita obter alta disponibilidade em sistemas de arquivos já existentes. Exemplo disso é o resultado que pôde ser obtido com o sistema de arquivos ext2, que pôde ser reimplementado como sendo o ext3 [TWE 2000].

Quanto aos tempos de recuperação, para sistemas de arquivos baseados em *journaling* eles são bastante reduzidos. A tarefa de verificação consiste apenas em inspecionar as transações pendentes do *log* - ao invés de percorrer todos os blocos buscando inconsistências. Com isso, na ocorrência de defeitos, o sistema de arquivos pode ser levado a um estado consistente pela aplicação das transações pendentes no *log* - ao invés de ser necessário inspecionar toda uma unidade de disco com o *fsck*.

3.1. O sistema de arquivos XFS

O XFS foi criado ainda em 1994 pela SGI para substituir o EFS [XFS 2004]. Ele é o mais antigo entre os sistemas de arquivos conhecidos, e a ênfase do seu projeto foi na capacidade de trabalhar com arquivos bastante grandes - da ordem de "*terabytes*". O XFS pode trabalhar usando um tamanho de bloco variando entre 512 bytes e 64 kbytes, provendo o suporte para sistemas de arquivos distribuídos (incluindo NFS versão 3), ACLs no padrão POSIX 1003.e, e quotas para usuários e grupos.

O estado de um sistema de arquivos XFS é o resultado da combinação de informações localizadas em três locais diferentes: disco, memória e *log* do *journaling*. O sistema de arquivos XFS somente estará em um estado consistente após ele ter sofrido *shutdown*, onde todos os dados residentes em memória (*buffers* e *cache*) são gravados em disco e as entradas do *log* de transações são aplicadas no sistema de arquivos.

4. Injeção de falhas no XFS

Qualquer sistema computacional é construído baseado em um modelo de falhas, e para essas falhas é que são definidos e implementados mecanismos de detecção de erros ou recuperação. Para a realização de um experimento de injeção de falhas, faz-se necessária a definição de um modelo de falhas que seja tão próximo quanto possível das falhas reais a que estará sujeito o sistema alvo durante operação em ambiente de produção. Com base no modelo de falhas são definidos e armazenados os cenários de falhas para um experimento. Esses cenários permitem repetir um experimento tantas vezes quantas forem necessárias, emulando operação durante um período de tempo compatível com o necessário para obter a medida de dependabilidade definida pelo usuário. Para este artigo, o objetivo dos experimentos de injeção de falhas é determinar a cobertura da detecção das falhas que possam ocorrer durante a utilização de um sistema de arquivos baseado em *journaling* e o tempo médio de recuperação do sistema após falha.

A maioria dos mecanismos de tolerância a falhas disponíveis nos sistemas de arquivos baseados em *journaling* consideram principalmente as falhas de hardware transientes (mais especificamente de

falta de energia), onde assume-se um modelo de *crash* para o sistema. Isto representa uma situação em que o sistema de arquivos estaria sendo utilizado e ocorreria a falha, fazendo com que a máquina fosse reiniciada sem que ocorresse um *flush* das informações em memória (tais como *buffers* e *cache*) ou das transações no *log* (elementos que, conforme a Seção 3.1, definem o estado do sistema de arquivos). Não estão previstos no modelo outros tipos de falhas comuns - como alteração de conteúdo de memória ou corrupção de disco.

Este modelo de falhas define o cenário de falhas que é gerado pela ferramenta FIJI. FIJI injeta falhas de *crash* de processos para determinar a cobertura dos mecanismos de recuperação e medir o tempo médio de recuperação. Entretanto, o modelo vai além disso pois propõe-se a avaliar o comportamento do XFS sob falhas do meio de armazenamento - frequentes em disco - que possam provocar a corrupção dos *logs*.

5. Arquitetura para injeção de falhas

O injetor de falhas FIJI caracteriza um ambiente de falhas como definido na Figura 1 derivado de arquitetura proposta por Hsueh [HSU 97]. O sistema alvo, ou seja o sistema sob teste por injeção de falhas, é uma máquina com uma partição Linux formatada com o utilitário mkfs.xfs [XFS 2004].

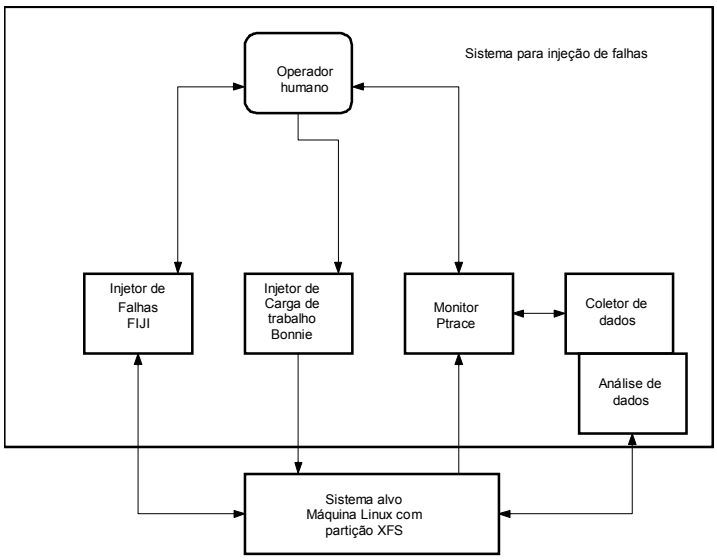

Figura 1 – Ambiente para injeção de falhas

O controle é exercido por um usuário, desenvolvedor ou avaliador de sistemas, que coordena a realização dos experimentos. A injeção de falhas é executada pela ferramenta FIJI, simulado a ocorrência de falhas do tipo *crash* e corrupção de meio magnético.

O utilitário *bonnie* atua como gerador de carga de trabalho, que deve gerar um *workload* correspondente às aplicações reais - tais como arquivos muito grandes (mais de 130 Mb), arquivos pequenos (menos de 600 Kb), servidor de arquivos, servidor de email, etc.

6. FIJI

A ferramenta FIJI (***F**ault **I**Njector for **J**ournaling f**I**lesystems*), desenvolvida no PPGC da UFRGS, é baseada nos recursos de depuração do sistema operacional Linux (os quais permitem a interceptação e manipulação de chamadas de sistema). Estes recursos foram utilizados para a implementação de um injetor de falhas específico para sistemas de arquivos baseados em *journaling*. Manipular os parâmetros de chamadas de sistema (ou *system calls*) equivale a alterar as requisições feitas ao sistema operacional.

Conceitualmente, o FIJI localiza-se entre a aplicação alvo e o sistema operacional. As falhas injetadas pelo FIJI estão de acordo com o modelo de falhas descrito anteriormente, e encontram-se especificadas no código-fonte da própria ferramenta. O FIJI é implementado como um processo

concorrente à aplicação alvo, usando os recursos oferecidos pelo sistema operacional Linux através da chamada de sistema *ptrace* [BAR 2000]. A chamada de sistema *ptrace* permite executar um processo de três maneiras: passo-a-passo, usando *breakpoints* ou executá-lo até a próxima chamada de sistema.

Para que seja possível controlar a execução de um processo com *ptrace*, é preciso que o FIJI esteja conectado ao processo. A funcão *ptrace* permite que isso seja conduzido de duas formas [LAD 98]:

- Conectando-se a um processo em execução, por meio da requisição PTRACE_ATTACH; ou
- Ativando-se o *flag* de depuração do processo antes de executá-lo, da seguinte forma: quando duplica-se um processo via *fork*, ativa-se o *flag* de depuração via uma requisição PTRACE_TRACEME.

A partir do momento em que o FIJI é executado, ele cria uma cópia de si mesmo através da função *fork* [TAN 97]. Neste novo processo gerado (processo filho) ativa-se o *flag* de depuração chamando-se a função *ptrace* com a requisição PTRACE_TRACEME descrita anteriormente. A seguir, o gerador de carga de trabalho, no caso o *bonnie*, é executado por meio da função *execve*. Em função de ter o *flag* de depuração ativado, na primeira chamada de sistema encontrada o processo filho pára e sinaliza o FIJI. Enquanto isso, o FIJI espera que o processo filho (já executando comandos) indique estar pronto para ser depurado. A espera é realizada com o auxílio da função *wait4*.

O FIJI executa uma requisição ao sistema operacional via função *ptrace* e espera, via função *wait4*, que o gerador de carga de trabalho sinalize o fim da execução. O sinal enviado, neste caso, é o SIGTRAP.

O FIJI recebe diretamente do processo filho a indicação de que está parado. Após receber o sinal de que o gerador de carga de trabalho encontrou uma chamada de sistema e parou, o FIJI pode requisitar ao sistema operacional que leia ou escreva na memória e nos registradores.

Agora, o FIJI pode inspecionar e manipular a memória do processo, o valor de seus registradores e a sua estrutura *user*. Isto é feito executando-se a chamada da função *ptrace* com o parâmetro PTRACE_GETREGS.

Por meio da chamada da função *ptrace* com o parâmetro PTRACE_GETREGS, é possível obter o valor de todos os elementos descritos na Figura 2. O trecho de código que segue foi obtido a partir do código-fonte para o *kernel* versão 2.6.6 para o sistema operacional Linux, no arquivo /usr/src/linux/include/asm/user.h.

```
struct user_regs_struct {
       long ebx, ecx, edx, esi, edi, ebp, eax;
       unsigned short ds, __ds, es, __es;
       unsigned short fs, __fs, gs, __gs;
       long orig_eax, eip;
       unsigned short cs, __cs;
       long eflags, esp;
       unsigned short ss, __ss;
};
```

Figura 2 – Elementos acessíveis à chamada de sistema *ptrace*

Assim que o FIJI lê o valor dos registradores, ele faz um teste para verificar se o conteúdo do registrador EAX é 03 (o que corresponde a uma chamada de sistema *read*) ou 04 (o que corresponde a uma chamada de sistema *write*) [BAR 2000]. Caso seja uma dessas, ele modifica o conteúdo do registrador EDX – o qual contém o tamanho do *buffer* utilizado. Depois disso, o FIJI executa novamente a função *ptrace* com o parâmetro PTRACE_SETREGS para zerar o registrador EDX e continuar executando a *syscall* com este parâmetro modificado. Zerando o registrador EDX, a operação de leitura ou escrita torna-se nula – mantendo o meio de armazenamento inalterado, segundo o modelo de falhas.

Apesar de haver amplas possibilidades para injeção de falhas, visto que podem ser alterados o conteúdo de registradores e posições de memória através da função *ptrace*; este recurso deve ser usado com moderação. Isso é necessário porque pedir a um processo que execute até um ponto de parada exige um custo computacional elevado em chaveamentos de contexto.

Existe um parâmetro, que deve ser informado para o FIJI pela linha de comando, informando dentro de quantos segundos o log será descartado e as chamadas de sistemas começarão a ser suprimidas. O descarte de log do journaling permite simular uma situação em que houve uma corrupção no meio de armazenamento. Para o modelo de falhas definido, basta realizar estas ações para comprometer a integridade do sistema de arquivos XFS.

7. Teste da ferramenta

Nesta seção são apresentados os testes inicias da ferramenta, e a comprovação da injeção de falhas realizada segundo o modelo de falhas proposto. Com o uso de um programa exemplo (cujo código-fonte está na Figura 3), é feita a injeção de falhas e os resultados são comprovados inspecionado um arquivo de saída.

Visando o teste da ferramenta FIJI, foi criada uma aplicação que armazena seqüências de caracteres em um arquivo. A seqüência e o resultado final de uma execução desta aplicação já são conhecidos. Alterações no tamanho do arquivo gerado ou em seu conteúdo significam que o injetor está agindo sobre a aplicação.

```
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
main()
{
       int fd;
       int i,c;
       char * msg1="AAAAAAA\n";
       char * msg2="BBBBBBB\n";

       if ((fd=open("arq_test.txt", O_SYNC | O_RDWR | O_CREAT, S_IRWXU |
S_IRWXG | S_IROTH)) < 0) {
              printf("Erro ao abrir o arquivo");
              exit(1);
       }
       for (i=0; i<5000; i++) {
              c = write(fd, msg1, 8);
              c = write(fd, msg2, 8);
       }

       close(fd);
       return(0);
}
```

Figura 3 – Listagem do programa para teste do FIJI

A metodologia de testes empregada é como segue: tomando-se este programa, com o resultado de execução já conhecido, pode-se injetar falhas e observar o resultado da execução. Como o roteiro de falhas é previamente definido, o resultado da execução – desde que o injetor funcione corretamente – também é conhecido.

Ao fim de uma execução correta do programa exemplo, existe um arquivo chamado arq_test.txt com 80.000 *bytes*. Ele contém seqüências alternadas de frases “AAAAAAA” e “BBBBBBB” – sempre seguidas por um avanço de linha (caractere “\n”) – vide Figura 4.

Depois de compilado este programa, a execução do mesmo apresenta o resultado da Figura 4:

```
[root@mail root]# gcc prog_test.c -o prog_test
[root@mail root]# ./prog_test
[root@mail root]# ls -l arq_test.txt
-rwxr-xr--    1 root     root        80000 Abr 30 08:53 arq_test.txt
```

Figura 4 – Resultado da execução do programa de teste do FIJI

No teste para injeção de falhas, especificou-se que 3 segundos após o início do processo, o *log* deveria ser destruído e as chamadas de sistema *read/write* descartadas. Uma nova execução do programa, agora com injeção de falhas, fornece como resultado a saída que está na Figura 5:

```
[root@mail root]# ./fiji -C 3 ./prog_test
...
[root@mail root]# ls -l arq_test.txt
-rwxr-xr--    1 root     root         1616 Abr 30 09:06 arq_test.txt
```

Figura 5 - Resultado da execução do programa de teste com o FIJI

Os defeitos provenientes dos erros injetados no experimento, observados no resultado da execução do programa exemplo, são condizentes com as regras de injeção de falhas consideradas. Desta forma, o funcionamento da ferramenta FIJI foi comprovado.

8. Conclusões

Atualmente estão sendo conduzidos experimentos para validar experimentalmente sistemas de arquivos baseados em *journaling* como o XFS e consolidar o ambiente proposto de acordo com uma metodologia desenvolvida pelo grupo de tolerância a falhas da UFRGS e aplicada anteriormente com sucesso em validação de mecanismos de recuperação em banco de dados.

Os experimentos estão direcionados para XFS em Linux, mas espera-se que a metodologia seja suficientemente genérica e portável para permitir a validação de outros sistemas de arquivos baseados em *journaling*. Testes preliminares comprovam a adequação da ferramenta ao modelo de falhas.

Os experimentos já realizados no XFS usando FIJI comprovam a eficiência do mecanismo de *journaling* para *crash* de processo. No momento estão sendo conduzidos os experimentos para avaliação do comportamento do XFS sob falhas do meio de armazenamento.

9. Referências bibliográficas

[BAR 2000] BAR, Moshe. **Linux Kernel Internals**. New York: McGraw-Hill, 2000, 351 p.

[HSU 97] HSUEH, Mei-Chen; TSAI, Timothy K.; IYER, Ravishankar K. Fault-Injection Techniques and Tools. **Computer**, v. 30, n. 4, Apr 1997, p.75-82.

[LAD 98] JOHNSON, Michael K. **Linux application development**. [S.l.]:Addison-Wesley, 1998. 538 p.

[SEL 2000] SELTZER, M. I.; et. al. *Journaling* versus soft-updates: Asynchronous meta-data protection in file systems. In: USENIX ANNUAL TECHNICAL CONFERENCE, 2000. **Proceedings...** San Diego, California, USA, 2000. Disponível em: < http://www.lcs.ece.cmu.edu/~soule/papers/seltzer.pdf>T. Acesso em: ago. 2003.

[TWE 2000] TWEEDIE, Stephen C. Ext3, *Journaling* Filesystem for Linux. In: OTTAWA LINUX SYMPOSIUM, 2000. **Proceedings...** Disponível em: <http://olstrans.sourceforge.net/release/OLS2000-ext3/OLS2000-ext3.html>. Acesso em: out. 2003

[XFS 2004] Sistema de arquivos XFS para Linux. Disponível em: <http://xfs.sourceforge.net>. Acesso em: abr. 2004.