Experiência de Injeção de Falhas no DataNode do Apache Hadoop

Eduardo Gondim[1], **Patrícia Pitthan Barcelos**[1], **Andrea S. Charão**[1]

[1]Laboratório de Sistemas de Computação
Universidade Federal de Santa Maria – UFSM

{egondim, pitthan, andrea}@inf.ufsm.br

Resumo. *Este trabalho descreve uma experiência de injeção de falhas em um dos componentes da plataforma Apache Hadoop. Esta plataforma é amplamente utilizada para processamento distribuído de grandes volumes de dados e dispõe de alguns mecanismos de tolerância a falhas. Com a experiência realizada, pôde-se analisar o comportamento do Hadoop em uma situação de falha não prevista nos testes incluídos na plataforma.*

1. Introdução

A plataforma Apache Hadoop é desenvolvida em Java e baseia-se em dois componentes principais: o Hadoop Distributed File System (HDFS) [ASF 2011a], que provê uma camada de armazenamento distribuído, e o Hadoop MapReduce [ASF 2011b], que provê uma camada de execução distribuída. Tanto o HDFS como o MapReduce foram projetados para suportar e tratar falhas nos nodos que compõem o ambiente de execução. Neste contexto, surge a necessidade de teste e validação de tais mecanismos.

A injeção de falhas é uma técnica de validação experimental amplamente usada no teste de mecanismos de tolerância a falhas. Este trabalho utiliza a injeção de falhas para analisar o comportamento mediante falhas de um dos componentes do HDFS, chamado DataNode. A experiência realizada emprega um *framework* de injeção de falhas que está disponível no próprio Hadoop, mas é pouco utilizado nos testes da plataforma. O restante deste artigo está organizado como segue: a seção 2 apresenta o Apache Hadoop, descrevendo sua arquitetura e seus mecanismos de tolerância a falhas. A seção 3 traz considerações sobre a injeção de falhas. Na sequência, a seção 4 apresenta o experimento realizado e a seção 5 faz considerações finais sobre o trabalho.

2. Apache Hadoop

A plataforma Hadoop segue uma arquitetura mestre-escravo, em que cada nodo do ambiente distribuído executa alguns processos para ativar funcionalidades do HDFS ou MapReduce. Estes processos são o NameNode, o DataNode, o JobTracker e o TaskTracker. O NameNode e o JobTracker executam em um nodo mestre, enquanto os processos DataNode e JobTracker executam principalmente em nodos escravos.

O NameNode, servidor mestre do HDFS, é responsável por gerenciar o espaço de nomes do sistema de arquivos e regular o acesso a arquivos pelos clientes. Quando uma requisição de abertura de arquivo chega, o NameNode informa ao cliente quais DataNodes possuem o arquivo e o cliente solicita diretamente ao DataNode.

O DataNode, por sua vez, gerencia o conteúdo do HDFS em cada nodo, armazenando blocos de dados no sistema de arquivos local e os metadados de cada bloco. Existe, portanto, um DataNode em cada nodo escravo do *cluster*. Se um DataNode falha, o NameNode manda os outros DataNodes que contêm cópia dos arquivos que estavam no DataNode falho se redistribuírem.

Um *cluster* Hadoop tem um único JobTracker em um nodo mestre e vários nodos escravos, cada um executando um processo TaskTracker. O JobTracker aceita submissões de *jobs* dos usuários (um *job* é um programa com tarefas *Map* e *Reduce*) e é responsável por escalonar as tarefas *Map* e *Reduce* aos TaskTrackers. Os TaskTrackers executam um *Map* ou um *Reduce*. O Jobtracker também valida o trabalho solicitado e, se ocorrer uma falha, escalona novamente a tarefa anterior.

3. Injeção de Falhas por Instrumentação de Código

Um dos mecanismos utilizados para injeção de falhas por software em tempo de execução é a instrumentação de código, também chamada de instrumentação dinâmica. Na instrumentação de código, as instruções são inseridas no programa alvo, permitindo que a injeção da falha aconteça antes, depois ou no lugar de instruções específicas.

A instrumentação de código permite que módulos de programas sejam completamente reescritos em tempo de execução, possibilitando a introdução de novas funcionalidades após a disponibilização da aplicação. Pode-se introduzir ou remover instruções, alterando o uso de classes, variáveis e constantes. Neste contexto, é possível realizar modificações antes de o código ser realmente executado.

Dentre as estratégias para a instrumentação de código pode-se citar a alteração direta do código, a reflexão computacional e a programação orientada aspectos (POA). Na alteração direta do código, o código instrumentado é espalhado na aplicação sob teste. Cada introdução de código implica em recompilação. Esta é uma das estratégias mais utilizadas, embora apresente como desvantagem a possibilidade de alteração na semântica da aplicação sob teste. A instrumentação de código por reflexão computacional, por sua vez, permite a alteração das classes de uma aplicação independentemente da disponibilidade do código fonte.

Na instrumentação de código por POA pode-se interceptar e instrumentar elementos de classes, bastando o conhecimento de suas APIs. Além disso, não são necessárias alterações no código da aplicação sob teste. O uso de uma estratégia orientada a aspectos possibilita que os aspectos sejam carregados junto com a aplicação alvo (os aspectos e as classes são combinados em tempo de carga) e a instrumentação seja realizada em tempo de execução.

3.1. Apache Hadoop Fault Injection Framework

Este *framework* começou a ser distribuído no Hadoop em meados de 2010, com o objetivo de dar suporte ao desenvolvimento de código de injeção de falhas [ASF]. Para isso, o FI Framework baseia-se na ferramenta AspectJ, para programação orientada a aspectos em Java [Soares and Borba 2002]. A orientação a aspectos facilita a separação entre o código de injeção de falhas e os códigos originais que se deseja instrumentar.

As falhas emuladas com este *framework* são de natureza não-determinística, ou seja, não se sabe de antemão quando uma falha irá ocorrer. Além disso, a ocorrência de

falhas é regulada por um modelo probabilístico simples, em que se configura um percentual de probabilidade de uma falha ocorrer. Para garantir que uma falha ocorrerá durante uma execução, a documentação do *framework* recomenda configurar o percentual de probabilidade para 100%.

Para se desenvolver uma injeção de falha, deve-se criar um aspecto em AspectJ (arquivo com extensão .aj). Este aspecto deve importar a classe do modelo de probabilidade do *framework* (`org.apache.hadoop.fi.ProbabilityModel`) e definir onde e como a falha irá ocorrer. O local da falha é definido por um `pointcut` em AspectJ, que seleciona os pontos do código original que serão interceptados durante a execução. Para estes pontos do código são definidos *advices* em AspectJ, que especificam o código adicional a ser executado quando a interceptação ocorrer. Os exemplos de injeção de falhas do FI Framework geralmente usam o *advice* `before`, que executa imediatamente antes da execução do ponto de código selecionado. A falha em si é especificada nesta parte do aspecto, e geralmente consiste no lançamento de uma exceção.

4. Experimentação

A experimentação descrita neste trabalho testa um modelo de falhas bastante amplo, o qual é utilizado apenas para observar o processo de defeito da aplicação alvo. É esperado que a aplicação fique em defeito quando sujeita a falhas que não está preparada para tolerar. A injeção deste tipo de falhas permite antecipar e avaliar o comportamento da aplicação, o qual sem injeção de falhas seria naturalmente difícil de conseguir.

Para o experimento foi utilizada a versão experimental e para desenvolvimento 0.21 do Hadoop. A realização se deu em duas etapas: primeiramente, selecionou-se um teste, incluído na plataforma Hadoop, para ser executado com e sem injeção de falhas. Em seguida, criou-se um aspecto, usando o FI Framework, para inserir uma falha que tivesse impacto no teste escolhido. O teste foi executado em um computador do Laboratório de Sistemas de Computação (LSC-UFSM), utilizando o modo pseudo-distribuído do Hadoop (uma só máquina executando todos os processos).

Para a seleção do teste, buscou-se um que usasse as diferentes classes do HDFS. Após análise de vários testes incluídos na plataforma, escolheu-se o programa `TestFileCreation.java`, que considera vários casos de teste durante a criação de um arquivo no HDFS. Neste programa, o arquivo é criado primeiramente no sistema de arquivos local, para então ser passado ao sistema de arquivos pseudo-distribuído. O arquivo é inserido no sistema conforme as regras de replicação do HDFS, sendo depois excluído, removendo-se também suas réplicas. Na sequência, o HDFS é terminado durante a exclusão das réplicas em cada DataNode. Há também uma parte do código que simula um erro na criação do arquivo e outra parte que testa o término abrupto do servidor HDFS, tendo o arquivo que ser fechado pelo HDFS. Em todos esses feitos pelo programa, é necessária a montagem do DataNode para que cada rotina fosse executada.

Analisando-se logs do TestFileCreation, decidiu-se focar a injeção de falhas na inicialização dos DataNodes. Para escolher o ponto de injeção, foi feita a análise do código em `DataNode.java`, que contém a classe de mesmo nome, e também de outros arquivos relacionados, buscando-se encontrar a linha de código que inicializa o DataNode. Em seguida foi criado um arquivo de aspecto sob nome `DataNodeStart.aj`, na pasta de testes que reúne todos os arquivos "`.aj`", que são mesclados com o código original.

Buscando emular uma falha externa ao software, o aspecto adiciona uma exceção no código sinalizando uma falha de memória (`OutOfMemoryError`). Com esta falha injetada, notou-se que o Hadoop, ao tentar iniciar um novo DataNode, não lida corretamente com tal erro, ocasionando uma falha generalizada que não é devidamente tratada no código de tolerância a falhas, ocasionando a interrupção da aplicação.

Foi realizada a análise do log gerado pelo teste com erro visando validar a inserção do código falho. Conforme o log, após o NameNode ser colocado no ar, é chamada a rotina que inicia os DataNodes. Esta então falha e os testes continuam tentando executar as operações no DataNode, que não existe, e com isso os erros se acumulam. Quanto ao tempo, notou-se que um teste normal do TestFileCreation era executado em aproximadamente 52 segundos. Com o teste falho não se passava de 1.7 segundos, uma vez que o erro acontecia na estrutura básica que dava suporte a ele.

5. Considerações Finais

Neste trabalho, analisou-se o comportamento do HDFS, mais precisamente do DataNode, conforme era simulada uma falha de memória na máquina. Através desta análise, foi possível observar que não há um tratamento adequado por parte do sistema de tolerância a falhas do Hadoop quando ocorre um erro na memória, o que pode ser comum em operações de larga de escala. Esta experiência, realizada com o auxílio do FI Framework do Hadoop, serve de exemplo para o desenvolvimento de novas falhas conforme aplicação específica.

Referências

ASF. Fault injection framework and development guide. Disponível em: `http://hadoop.apache.org/hdfs/docs/current/faultinject_framework.html`. Acesso em: 19 dezembro 2011.

ASF (2011a). Welcome to Hadoop Distributed File System! Disponível em: `http://hadoop.apache.org/hdfs`. Acesso em: 19 dezembro 2011.

ASF (2011b). Welcome to Hadoop MapReduce! Disponível em: `http://hadoop.apache.org/mapreduce`. Acesso em: 19 dezembro 2011.

Soares, S. and Borba, P. (2002). *AspectJ – Programação orientada a aspectos em Java*. Disponível em: `http://www.cin.ufpe.br/~phmb/papers/AspectJTutorialSBLP2002.pdf`. Acesso em: 20 julho 2011.