UNIVERSIDADE ESTADUAL DE CAMPINAS
INSTITUTO DE COMPUTAÇÃO

Projeto de Dissertação de Mestrado: Uma Solução de Alta Disponibilidade para o Sistema de Arquivos Distribuído do Hadoop

Orientando: André Oriani
Orientadora: Prof.ª Dr.ª Islene Calciolari Garcia

Resumo

Sistemas de arquivos distribuídos tem por objetivo armazenar grandes volumes de dados com consistência e fornecê-los com alta disponibilidade. Eles cumprem muito bem esse objetivo por meio do emprego de técnicas tais como arranjos redundantes de discos independentes, replicação de arquivos e somas de verificação. No entanto, a disponibilidade como um todo desses sistemas ainda é questionável. Importantes representantes desta classe de sistemas de arquivos concentram o gerenciamento de todo o espaço de nomes, ou pelo menos parte considerável dele, num único processo. O mesmo processo também é responsável por fazer a interface com os usuários. Consequentemente falhas neste processo tem o potencial de comprometer todo o sistema, o que pode ser indesejável para aplicações interativas ou de missão crítica. Esse projeto de Mestrado tem por foco propor um solução para este problema, conferindo alta disponibildade ao Hadoop Distributed File System.

1 Introdução

O volume de dados produzido diariamente nunca foi tão grande. Para se ter uma ideia de tal volume basta dizer que o Google processa diariamente cerca de 20 petabytes [33]. O Grande Colisor de Hádrons do CERN tem expectativa de gerar 15 petabytes de dados por ano [40]. O sequenciamento do genoma de um único ser humano é capaz de produzir 90 gigabytes de dados brutos [37]. E por fim, o projeto SETI estima que uma varredura completa do céu em busca de sinais extraterrestres gere 39 terabytes de dados [25].

Porém, tamanha quantidade de dados é inútil se não puder ser analisada rapidamente e transformada em informação. A solução mais barata, eficiente e escalável para esta tarefa complexa é a computação distribuída. O Google, por exemplo, alegava em 2003 utilizar um *cluster* com mais de 15 mil máquinas para atender às milhares de consultas que recebia por segundo [1]. E uma única consulta demandava em média o acesso a centenas de megabytes de dados e o consumo de dezenas de bilhões de ciclos de CPU. Em 2008, o Yahoo anunciou o lançamento de um *cluster* composto por mais de 10 mil cores para executar seu sistema de indexação de páginas [41]. O sistema analisava por volta de um trilhão de links para produzir mais de 300 terabytes de informação comprimida.

Atualmente um dos mais conhecidos frameworks para processamento distribuído é o Hadoop MapReduce, um dos componentes do projeto Hadoop[1] da Apache Software Foundation[2]. O projeto Hadoop tem por objetivo "*desenvolver software de código aberto para computação confiável, escalável e distribuída*" [28]. Aliás, o Hadoop MapReduce foi framework utilizado por Yahoo para construir o sistema de indexação citado no exemplo acima. Hadoop MapReduce é uma implementação livre do paradigma computacional para o processamento de grandes conjuntos de dados, introduzido por engenheiros de software do Google no artigo "*MapReduce: Simplified Data Processing on Large Clusters*" [9]. MapReduce divide a entrada em vários agrupamentos pequenos e independentes que são processados em paralelo por vários processos denominado *mappers*. As saídas dos *mappers* são então combinadas por processos *reducers* para se chegar a resposta final. Obviamente para que o Hadoop MapReduce funcione adequadamente é necessário o apoio de um sistema de amazenamento distribuído que seja escalável e que garanta alta disponibilidade dos dados e uma boa vazão. O componente do Hadoop que cumpre esse papel é o Hadoop Distributed Filesystem (HDFS) [14, 26, 20].

O HDFS é capaz de armazenar 21 petabytes [32] distribuídos entre 4000 nós e servir 60 milhões de arquivos para 15 mil clientes [19]. A vazão do sistema pode atingir valores maiores que 40Mb/s tanto para leitura como escrita. Além do Hadoop MapReduce, o HDFS é também utilizado pelo HBase[3], outro componente do Hadoop que implementa um banco de dados distribuído, versionado e orientado a colunas. Há também relatos de uso do HDFS pela comunidade científica para armazenar resultados de experimentos [3] e de seu emprego em aplicações que variam desde servidores multimídias a redes de sensores [5]. Esta recente popularidade do HDFS como um sistema de armazenamento de propósito geral deve-se à sua confiabilidade e à sua capacidade de manter a consistência e a alta disponibilidade dos dados.

Assim como PVFS [7], Lustre [38], Panasas [24], Ceph [23] e Apple Xsan2 [30], o HDFS pertence à categoria de sistemas de arquivos distribuídos que mantém os dados ortogonais ao

[1] `http://hadoop.apache.org`

[2] `http://www.apache.org`

[3] `http://hbase.apache.org`

espaço de nomes. Nesses sistemas existem dois tipos de nós: servidores de metadados e servidores de armazenamento de objetos. Os servidores de metadados são responsáveis por gerenciar a árvore do sistema de arquivos bem como outros metadados (permissões, usuário, grupo, etc.). Também são responsáveis por atender requisições de clientes e por manter o registro da localização dos arquivos no sistema. Por outro lado, os servidores de armazenamento de objetos armazenam os dados dos arquivos. No caso específico do HDFS, por razões de simplificação de projeto, existe apenas um único servidor de metadados. Logo esse nó configura-se com um ponto único de falha do sistema, visto que seu desligamento por motivos de funcionamento anormal ou de manutenção implica na indisponibilidade de todo o sistema. Como cada vez mais o Hadoop MapReduce e outras aplicações que usam o HDFS estão sendo empregados em sistemas de missão crítica, esta limitação está se tornando intolerável.

Desta forma, o objetivo primordial desse projeto de mestrado é a remoção desse ponto único de falha por meio da replicação do servidor de metadados. Espera-se com isso conferir alta disponibilidade ao sistema.

O restante desse documento se organiza da seguinte forma: a Seção 2 descreve a arquitetura e funcionamento do HDFS; a Seção 3 mostra como a alta disponibilidade do servidor de metadados vem sendo tratada em outros sistemas de arquivos distribuídos e no próprio HDFS; a Seção 4 enumera as contribuições esperadas desse projeto; a Seção 5 discute desafios a serem enfrentados e, finalmente, a Seção 6 estabelece o cronograma do projeto.

2 Hadoop Distributed Filesystem

Conforme já mencionado anteriormente, o HDFS é um sistema de arquivos distribuído voltado para computação de alta performance, escalável e com ênfase na consistência e disponibilidade dos dados. Por ter sido inspirado no Google File System [12] e voltado inicialmente para dar apoio às tarefas do Hadoop MapReduce, o sistema apresenta as seguintes características:

- O HDFS é executado em *clusters* de centenas ou milhares de nós constituídos de *commodity hardware*, o qual possui baixo custo e pode ser fornecido por diversos fabricantes. Esse grande número de elementos envolvidos gera uma probabilidade não trivial para falhas. Consequentemente o HDFS faz uso intensivo de somas de verificação e replicação para detecção e mascaramento de problemas;

- O sistema é otimizado para arquivos grandes, na ordem de gigabytes ou terabytes de tamanho. Para armazenar arquivos pequenos de forma eficiente, o HDFS conta um funcionalidade parecida com o comando `tar` do Unix que armazena vários arquivos num só;

- O sistema suporta apenas um escritor, porém vários leitores simultaneamente. Dados somente podem ser adicionados ao final do arquivo. Ou seja, não são permitidas escritas aleatórias. Esse modelo de consistência simples porém restritivo diminui sensivelmente o esforço de implementação;

- Jobs do Hadoop MapReduce necessitem de leituras extensas em *streaming*. Por isso o HDFS favorece alta vazão em detrimento de baixa latência. O sistema de arquivos possui uma interface muito próxima do padrão POSIX, entretanto a observância a algumas especificações é relaxada para atingir o objetivo de alta vazão.

A atual arquitetura do HDFS é exibida na Figura 1. Ela segue um padrão de mestre-escravos. O mestre, o servidor de metadados é denominado namenode. Os escravos, os servidores de armazenamento de objetos, são chamados datanodes. O papel do backupnode será discutido posteriormente.

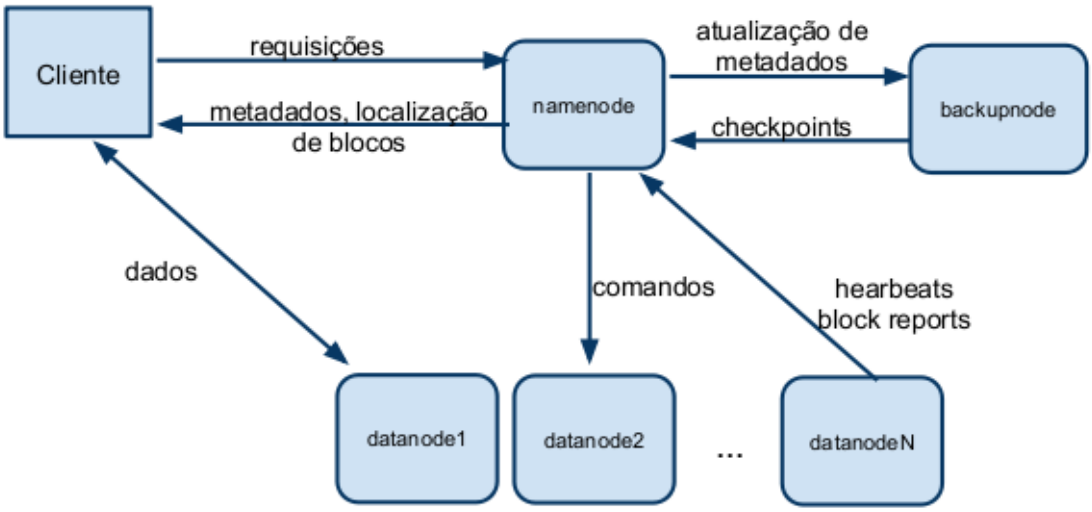

Figura 1: Arquitetura do HDFS e mensagens trocadas entre nós.

Assim como em sistemas de arquivos convencionais, arquivos no HDFS são compostos por blocos de tamanho fixo. Os datanodes são os responsáveis por armazenar os blocos, os quais são representados localmente por dois arquivos. Um armazena os dados do bloco, enquanto o outro guarda seus metadados tais com identificação do bloco, versão e somas de verificação. Os blocos possuem um tamanho grande, com valores típicos de 64Mb e 128Mb. A razão para isso é que se o tamanho do bloco for tal que tempo de sua transferência for consideravelmente maior que o tempo de *seek* do disco, a transferência do bloco pode ocorrer na mesma taxa de transferência do disco. Blocos são replicados, o que garante proteção contra perda de dados e boa performance de leitura, visto que várias réplicas podem ser lidas em paralelo. O nível de replicação, que por padrão é três, pode ser definido individualmente para cada arquivo. O algoritmo padrão de posicionamento de réplicas tenta garantir um bom *trade-off* entre confiabilidade e desempenho de leitura e escrita. Pelo menos duas réplicas são postas no mesmo *rack* mas em nós diferente, e pelo menos outra réplica é posta em um outro *rack* distinto.

O namenode é o responsável por gerenciar a árvore de diretórios e arquivos, usando estruturas similares a inodes. Desta forma ele guarda informações como o nome, caminho, tempo de acesso e modificação, permissões, dono, grupo, cotas e a relação de blocos que compõe o arquivo. Periodicamente, os datanodes enviam mensagens de *heartbeat* para informar o namenode de que estão ativos e de suas respectivas capacidades livres atuais. Também enviam, mas com uma periodicidade menor, *block reports*, que são relatórios de todos os blocos que o datanode é capaz de oferecer. Com isso, o namenode consegue ter uma visão global da localização e do nível de replicação de cada bloco no sistema. Caso o nível de replicação de um bloco não esteja de acordo com o que foi definido, o namenode instrui datanodes a copiarem réplicas entre si, por meio de comando enviados em resposta às mensagens de *heartbeat*. Por questões de performance,

o namenode mantém todo seu estado em memória principal. Para evitar que todo o sistema de arquivos seja perdido em caso de falha do namenode, uma combinação de *checkpoints* e *logs* transacionais é empregada para tornar a recuperação possível. Assim, operações que modifiquem o espaço de nomes são registradas em memória secundária estável. Ao iniciar, o próprio namenode cria um novo *checkpoint* a partir do fusão do *log* com o *checkpoint* anterior. Durante a execução, um segundo nó, o backupnode, fornece *checkpoints* adicionais para o namenode, uma vez que o backupnode recebe constantemente notificações de alterações no espaço de nomes. Informações relativas à localização das réplicas não são persistidas em *checkpoints* ou *logs* por serem voláteis e mudarem muito rapidamente. Desta forma, quando o namenode inicia, ele espera por *block reports* da maioria dos datanodes para reconstruir o mapeamento entre blocos e datanodes. Só quando este mapeamento está reconstruído é que o namenode passa à atender as requisições dos clientes.

Os usuários do HDFS interagem com o sistema por meio de um cliente cuja API se assemelha com a interface definida pelo padrão POSIX. A exemplo de toda comunicação entre nós, o cliente se comunica com o namenode e datanodes por meio de chamadas de procedimento remoto. Para leituras, o cliente solicita ao namenode a lista de blocos do arquivo de interesse bem como os datanodes que contém as réplicas dos blocos. O cliente pode então ler dos datanodes mais próximos, garantido assim uma boa vazão. Para escritas, o cliente solicita que o namenode aloque um conjunto de datanodes para armazenar as réplicas. Durante a escrita, os datanodes são organizados em um *pipeline*, com um datanode repassando para o seguinte os dados a serem escritos. Conforme já foi dito, apenas um escritor por arquivo é permitido e o controle disso é feito por um mecanismo de *leasing*. Um processo escritor deve solicitar uma *lease* e renová-la periodicamente para pode continuar a escrever. Do contrário a *lease* pode expirar e ser concedida a um novo escritor. É importante salientar que todo tráfego de dados ocorre diretamente entre cliente e datanodes, sem intermediação do namenode, de forma que este último não se torne um gargalo.

3 Trabalhos relacionados

A alta disponibilidade do servidor de metadados de sistemas de arquivos distribuídos é ainda um campo em desenvolvimento. Para ilustrar tal afirmação, os parágrafos a seguir descrevem como diferentes sistemas distribuídos lidam com a questão.

O Parallel Virtual File System (PVFS) [7] é sistema de arquivos paralelo de código aberto e desenvolvido em conjunto por laboratórios de pesquisa e universidades estadunidenses. A documentação do PVFS [6] recomenda o uso do gerenciador de recursos de cluster Hearbeat do projeto Linux-HA [36]. O Hearbeat monitora continuamente o servidor de metadados. Caso detecte a falha do servidor atual, o Heartbeat inicia outra instância automaticamente. Esse processo de iniciar uma nova instância do serviço constatada a queda da instância atual é conhecido na literatura por *failover*.

Lustre [38], um outro sistema de arquivos distribuído de código aberto comercializado pela Oracle, emprega um estratégia de par ativo-passivo. O par compartilha um dispositivo de armazenamento no qual o *log* transacional é escrito. Caso o servidor de metadados passivo detecte a falha no ativo, ele assume a posição de ativo a partir do *log* do dispositivo compartilhado.

Panasas [24] divide o espaço de nomes em volumes, cada um mantido por seu próprio servidor de metadados. Os servidores também usam a estratégia de *journaling* e encaminham o *log* transacional para um par passivo remoto que pode assumir em caso de falhas.

Ceph de Weil et al [23], dinamicamente particiona a árvore do sistema de arquivos entre vários servidores de metadados conforme a demanda por arquivos. Sub-árvores populares podem ser replicadas em vários servidores. Embora seja usado *journaling*, não há um mecanismo de *failover* a partir do *log* transacional como nos dois últimos exemplos. Embora o particionamento do espaço de nomes vise aumentar a escabilidade do sistema, ele também ajuda a restringir o impacto de falhas nos servidores de metadados

O Xsan2 da Apple [30], também divide o espaço de nomes em volumes. E também cada volume é gerenciado por seu próprio servidor de metadados. Cada servidor de metadados pode ter vários servidores passivos, com diferente prioridades atribuídas a eles. O servidor passivo de maior prioridade assume em caso de queda do servidor de metadados ativo.

Como último exemplo citamos o Google File System (GFS) [12] que inspirou o HDFS. O GFS também usa *journaling*, com *checkpoints* e *logs* sendo replicados para várias máquinas. Caso a infraestrutura de monitoriamento, que é externa ao GFS, detecte falha no servidor de metadados, ela pode iniciar um novo em qualquer parte do sistema a partir do *log*. O Google File System também possuem servidores que podem responder a requisições de somente leitura de clientes. Porém, eles podem estar defasados em relação ao estado atual do servidor de metadados.

Obviamente, o HDFS sendo um projeto de software livre do qual também participam grandes empresas como Yahoo e Facebook (as quais o usam ativamente) também recebeu propostas de como o disponibilidade do namenode poder ser melhorada. A mais simples de todas [2] e que não exige mudanças de código é semelhante à estratégia adotada pelo Lustre. Porém, ao invés de usar armanezamento compartilhado, a solução usa DRBD [31] que poder ser sucintamente descrito como "RAID nível 1 baseado em rede". Em outras palavras, DRDB garante que os discos do namenode ativo e do passivo estejam sincronizados, de forma que o passivo tenha a versão mais recentente do *checkpoint* e do *log* transacional quando assumir. Assim como o PVFS, essa solução também usa Heartbeat para detectar a falha no namenode ativo e iniciar o processo de *failover* .

Como mencionado na seção anterior, o backupnode recebe notificaçõoes sobre todas as mudanças ocorridas no espaço de nomes, de que forma que ele fique sincronizado com o namenode. Engenheiros do Yahoo o projetaram assim para que no futuro fosse implementado um mecanismo de *failover* que permitisse ao backupnode tomar o lugar do namenode [39]. Porém como o backupnode não possui alguma informação sobre a localização do blocos, esse mapeamento teria de ser reconstruído tal como é feito na iniciação do namenode. Enquanto o mapeamento não é restaurado, requisições de clientes não podem ser atendidas.

China Mobile Research [22] e IBM China Research [21] tiveram idéias semelhantes. Ambas adotam um arquitetura ativo-passivos. O namenode ativo é definido por meio de um algoritmo de eleição. O namenode ativo além de atender os clientes, ocupa-se também de enviar atualizações de estado para os demais namenodes. A diferença principal entre as duas soluções é que na solução da IBM apenas atualizações no espaço de nomes e *leasing* são propagadas. Já a solução da China Mobile também envia informações sobre *heartbeats* e *block reports*, o que lhe permite ter um tempo menor para a duração do *failover*. A desvantagem desse tipo de solução é que como o namenode ativo é responsável por atualizar cada um dos namenodes passivos de forma praticamente síncrona, o *lock* do HDFS é mantido por um tempo maior, resultando numa sensível degradação da performance do sistema.

Desenvolvedores do Facebook criaram uma solução que chamaram de AvatarNodes [4]. Essa solução também usa um par ativo-passivo. O namenode ativo grava seu *log* transacional numa partilha NFS. O namenode passivo lê continuamente o *log* da partilha e aplica as transações para

atualizar sua visão do espaço de nomes. Datanodes são modificados para enviar as mensagens de *heartbeat* e *block reports* para ambos os namenodes. Consequentemente o namenode também possui informação sobre a localização dos blocos, o que diminui o tempo do processo de *failover* . Nesta solução o *failover* é feito manualmente pelo administrador.

Clement et al desenvolveram a biblioteca UpRight [8] para adicionar proteção contra falhas bizantinas de maneira simples e viável a sistemas já tolerantes a falhas por queda. Para provar que atigiram seu objetivo, eles modificaram o HDFS para utilizar a biblioteca desenvolvida. Como resultado obtiveram a replicação ativa (ou replicação por máquina de estados) [18] do namenode, com requisições de clientes sendo ordernadas globalmente e validadas por meio de MACs (*message authentication codes*). Todavia, a pouca atenção que o atual HDFS dá à questão da segurança torna a proteção contra falha bizantinas algo supérfluo. O código da modificação foi baseado em uma versão antiga do HDFS e não é recomendado para uso em sistemas em produção [35].

Em todas a soluções de alta disponibilidade descritas aqui (exceto pela de Clement et al) ocorre o fenômeno de *failover*. Durante período de tempo em que ele ocorre o sistema fica indisponível. Consequentemente, requisições de clientes podem falhar ou não serem executadas. Logo, falhas no servidor de metadados não são completamente transparentes para os usuário do sistema. Isso pode gerar uma latência indesejável para aplicações interativas ou de missão crítica. Portanto existe a necessidade de se pesquisar soluções de alta disponibilidade para sistemas de arquivos distribuídos nas quais não hajam períodos de indisponibilidade, mesmo que pequenos, enquanto houver um número suficiente de réplicas.

4 Contribuições Esperadas

Espera-se obter como resultado desse projeto de mestrado uma solução de alta disponibilidade do HDFS usando replicação ativa (replicação por máquina de estados) do namenode. A solução deverá ser t-tolerante a falhas por queda, com $t > 1$ preferencialmente. Para tal será necessário a utilização de um protocolo de multicast atômico tal como Paxos [15] para que todas as réplicas do namenode recebam todas as requisições na mesma ordem. Também será necessário a transformação do namenode numa máquina de estados determinística, a fim de garantir consistência entre as réplicas. A preferência por replicação ativa se deve ao fato dessa estratégia não apresentar *failover*, o que é uma característica altamente desejável conforme foi exposto na seção anterior. Entretanto, peculiaridades do HDFS, que serão descritas na próxima seção podem moldar os rumos para uma solução híbrida entre replicação ativa e passiva [13].

Para a validação da implementação poderão ser usados os próprios testes de *benchmark* que vem com o HDFS. Isso permitirá avaliar os efeitos da solução proposta no desempenho do sistema. Também serão feitos testes que simulem falhas nas réplicas do namenode para verificar o comportamento da solução. Com isso poderemos constatar se o objetivo de prover alta disponibilidade foi atingido. No caso de optarmos por replicação passiva, será importante mensurar o tempo de *failover* . Como o HDFS é normalmente utilizado em ambientes com muitos nós (de centenas a milhares), pretende-se utilizar o serviço Amazon EC2[4] como ambiente de testes devido às facilidades que ele provê para criar várias instâncias de máquinas virtuais.

O código produzido será publicado sob a licença Apache versão 2 [29], a mesma do projeto Hadoop. Ele será disponibilizado ao público em geral por meio de algum serviço de hospedagem

[4] `http://aws.amazon.com/ec2/`

de repositório de código tal como o SourceForge[5] ou GitHub[6]. A disponibilização do código permite que ele seja estudado e reaproveitado por desenvolvedores de outros sistemas de arquivos distribuídos, de forma que ele seja uma contribuição real para a área. Pretende-se que ele seja incorporado de volta ao HDFS por meio do processo de HEP – Hadoop Enhancement Proposal (proposta de melhoria do Hadoop) [34]. A submissão de um HEP tem também por objetivo colher *feedback* da comunidade de usuários e desenvolvedores do Hadoop sobre o trabalho desenvolvido.

Por fim espera-se também a elaboração e publicação de artigos em periódicos e eventos relacionados, relatando as experiências com o projeto e os resultados obtidos.

5 Desafios

Para que replicação ativa funcione, faz-se necessário que o namenode se comporte de maneira determinística, pois só assim as réplicas progredirão segundo a mesma sequência de estados, garantindo consistência entre elas. Infelizmente a atual implementação do namenode é inerentemente não determinística. O tempo do relógio físico é usado para definir a duração das *leases* e para agendar tarefas de manutenção para o gerenciamento dos blocos. Números aleatórios são usados no algoritmo de posicionamento das réplicas. Estruturas de dados cuja iteração não é determinística são usadas para gerenciar os blocos. Para cada cliente ou datanode que se conecta ao namenode é criado uma *thread*. Também são criadas *threads* para monitorar *leases* e o nível de replicação de blocos. Todos esses fatores — uso do relógio físico, números, estruturas de dados não determinísticas e *multithreading* — corroboram para o não determinismo do namenode.

Obviamente alguns indeterminismos podem ser removidos. Pode-se fazer todas as réplicas usarem a mesma semente, adotarem relógios lógicos, etc. Todavia nem sempre isso será possível. A data de criação de um arquivo precisa ser obtida de um relógio físico e todas as réplicas precisam concordar com esse valor. Pode-se fazer o namenode usar uma única *thread* ou usar alguma extensão do ambiente Java [11] (o HDFS é implementado em Java) ou *frameworks* [16] que garanta escalonamento de *threads* e obtenção de *locks* de maneira determinística. Entretanto ambas abordagens são pouco práticas e o desempenho do sistema final pode ser sofrível.

Sendo assim talvez seja necessário adotar uma postura híbrida entre replicação ativa e passiva, já que a última é mais flexível com relação a indeterminismos. Um exemplo disso seria adotar a estratégia de deixar para a primeira réplica que executar a requisição definir quais serão os parâmetros não determinísticos que devem ser usados pelas outras réplicas. Isso de certa forma já é feito no HDFS com o *log* transacional. Embora o cliente fornece apenas dois parâmetros — o nome antigo e o novo — para renomear um arquivo, a operação é registrada no *log* com três parâmetros, sendo o parâmetro adicional o tempo de modificação do arquivo definido pelo relógio físico do namenode. Exemplos de estratégias híbridas podem ser vistas na literatura como em [27] e [10].

Outro desafio a ser considerado é a incorporação do código desenvolvido ao projeto Hadoop. Embora existam muitas pesquisas em torno do Hadoop, apenas uma pequena parcela delas é incorporada na base de código do projeto. A razão para isso é que a maioria delas é desenvolvida em completo isolamento, sem envolvimento da comunidade de desenvolvedores. Geralmente elas trabalham com um *baseline* de código fixo e seguem um caminho ignorando o *roadmap* do pro-

[5] `http://www.sf.net`

[6] `http://www.github.com`

jeto. Muitas vezes o código produzido por elas não tem qualidade suficiente para ser usado em aplicações da vida real. Pretende-se superar essa dificuldade por meio da experiência do candidato como engenheiro de software e por um trabalho em constante contato com a comunidade de desenvolvimento. No momento contamos com a colaboração de Rodrigo Schmidt, ex-aluno do programa de pós-graduação desse Instituto, participante ativo do projeto Hadoop e que sugeriu esse projeto de pesquisa.

6 Cronograma

Antes de ser aceito no Programa de Mestrado Acadêmico como aluno regular, o candidato cursou as seguintes disciplinas na condição de aluno especial de pós-graduação:

- MO633 - Banco de Dados II ;
- MO611 - Teleprocessamento e Redes;
- MO806 - Tópicos em Sistemas Operacionais: Programação multithread e outros tópicos avançados em sistemas operacionais;
- MO405 - Teoria dos Grafos;
- MO806 - Tópicos em Redes de Computadores I: Gerência de Redes .

Para a execução do projeto as seguintes atividades estão planejadas:

- **Atividade A** Disciplina MO901 - Seminários de Software Livre
- **Atividade B** Disciplina MO441 - Computação Distribuída
- **Atividade C** Elaboração do Relatório Técnico: *The Search for a Highly-Available Hadoop Distributed Filesystem* [17]
- **Atividade D** Estudo Dirigido : código do HDFS, Replicação e Multicast Atômico
- **Atividade E** Exame de Qualificação - preparação e defesa
- **Atividade F** Desenvolvimento iterativo da proposta de solução - pesquisa, projeto, codificação e testes unitários de forma iterativa.
- **Atividade G** Avaliação e validação da proposta de solução por meio de testes funcionais e de performance
- **Atividade H** Escrita da Dissertação de Mestrado
- **Atividade I** Defesa de Mestrado

2010

Atividade	Jan	Fev	Mar	Abr	Mai	Jun	Jul	Ago	Set	Out	Nov	Dez
Atividade A												
Atividade B												
Atividade C												
Atividade D												
Atividade E												
Atividade F												

2011

Atividade	Jan	Fev	Mar	Abr	Mai	Jun	Jul	Ago	Set	Out	Nov	Dez
Atividade F												
Atividade G												
Atividade H												

2012

Atividade	Jan	Fev	Mar	Abr	Mai	Jun	Jul	Ago	Set	Out	Nov	Dez
Atividade H												
Atividade I												

Referências

[1] Barroso, L. A., Dean, J., and Hölzle, U. Web search for a planet: The google cluster architecture. *IEEE Micro 23*, 2 (2003), 22–28.

[2] Bisciglia, C. Hadoop HA configuration. `http://www.cloudera.com/blog/2009/07/hadoop-ha-configuration/`. Último acesso em 17 de agosto de 2010.

[3] Bockelman, B. Using Hadoop as a grid storage element. *Journal of Physics: Conference Series 180*, 1 (2009), 012047.

[4] Borthakur, D. Hadoop avatarnode high availability. `http://hadoopblog.blogspot.com/2010/02/hadoop-namenode-high-availability.html`. Último acesso em 17 de agosto de 2010.

[5] Borthakur, D. HDFS high availability. `http://hadoopblog.blogspot.com/2009/11/hdfs-high-availability.html`. Último acesso em 17 de agosto de 2010.

[6] Borthakur, D. PVFS2 High-Availability Clustering using Heartbeat 2.0. `http://www.pvfs.org/cvs/pvfs-2-7-branch.build/doc/pvfs2-ha-heartbeat-v2/pvfs2-ha-heartbeat-v2.php`. Último acesso em 17 de agosto de 2010.

[7] Carns, P. H., Ligon, III, W. B., Ross, R. B., and Thakur, R. Pvfs: a parallel file system for linux clusters. In *ALS'00: Proceedings of the 4th annual Linux Showcase & Conference* (Berkeley, CA, USA, 2000), USENIX Association, pp. 28–28.

[8] Clement, A., Kapritsos, M., Lee, S., Wang, Y., Alvisi, L., Dahlin, M., and Riche, T. UpRight cluster services. In *SOSP '09: Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles* (New York, NY, USA, 2009), ACM, pp. 277–290.

[9] Dean, J., and Ghemawat, S. MapReduce: Simplified data processing on large clusters. In *OSDI* (2004), pp. 137–150.

[10] Defago, X., Schiper, A., and Sergent, N. Semi-passive replication. In *Reliable Distributed Systems, 1998. Proceedings. Seventeenth IEEE Symposium on* (20-23 1998), pp. 43 –50.

[11] Domaschka, J., Bestfleisch, T., Hauck, F. J., Reiser, H. P., and Kapitza, R. Multithreading strategies for replicated objects. In *Middleware '08: Proceedings of the 9th ACM/IFIP/USENIX International Conference on Middleware* (New York, NY, USA, 2008), Springer-Verlag New York, Inc., pp. 104–123.

[12] Ghemawat, S., Gobioff, H., and Leung, S.-T. The Google file system. *SIGOPS Oper. Syst. Rev. 37*, 5 (2003), 29–43.

[13] Guerraoui, R., and Schiper, A. Software-based replication for fault tolerance. *Computer 30*, 4 (1997), 68–74.

[14] HDFS architecture. `http://hadoop.apache.org/common/docs/current/hdfs_design.html`. Último acesso em 17 de agosto de 2010.

[15] Lamport, L. The part-time parliament. *ACM Trans. Comput. Syst. 16*, 2 (1998), 133–169.

[16] Olszewski, M., Ansel, J., and Amarasinghe, S. Kendo: efficient deterministic multithreading in software. *SIGPLAN Not. 44*, 3 (2009), 97–108.

[17] Oriani, A., Garcia, I. C., and Schmidt, R. The Search for a Highly-Available Hadoop Distributed Filesystem. Tech. Rep. IC-10-24, Institute of Computing, University of Campinas, August 2010. In English, 27 pages.

[18] Schneider, F. B. Implementing fault-tolerant services using the state machine approach: a tutorial. *ACM Comput. Surv. 22*, 4 (1990), 299–319.

[19] Shvachko, K. HDFS scability: the limits to growth. *;login: The Usenix Magazine 35*, 2 (April 2010), 6–16.

[20] Shvachko, K., Kuang, H., Radia, S., and Chansler, R. The Hadoop Distributed File System. In *Mass Storage Systems and Technologies (MSST), 2010 IEEE 26th Symposium on* (3-7 2010), pp. 1 –10.

[21] Wang, F., Qiu, J., Yang, J., Dong, B., Li, X., and Li, Y. Hadoop high availability through metadata replication. In *CloudDB '09: Proceeding of the first international workshop on Cloud data management* (New York, NY, USA, 2009), ACM, pp. 37–44.

[22] Wang, X. Pratice of namenode cluster for HDFS HA. `http://gnawux.info/hadoop/2010/01/pratice-of-namenode-cluster-for-hdfs-ha/`. Último acesso em 17 de agosto de 2010.

[23] Weil, S. A., Brandt, S. A., Miller, E. L., Long, D. D. E., and Maltzahn, C. Ceph: a scalable, high-performance distributed file system. In *OSDI '06: Proceedings of the 7th symposium on Operating systems design and implementation* (Berkeley, CA, USA, 2006), USENIX Association, pp. 307–320.

[24] Welch, B., Unangst, M., Abbasi, Z., Gibson, G., Mueller, B., Small, J., Zelenka, J., and Zhou, B. Scalable performance of the Panasas parallel file system. In *FAST'08: Proceedings of the 6th USENIX Conference on File and Storage Technologies* (Berkeley, CA, USA, 2008), USENIX Association, pp. 1–17.

[25] Werthimer, D., Cobb, J., Lebofsky, M., Anderson, D., and Korpela, E. SETI@HOME—massively distributed computing for SETI. *Computing in Science and Engg. 3*, 1 (2001), 78–83.

[26] White, T. *Hadoop: The Definitive Guide*. O'Reilly Media Inc, Sebastopol, CA, June 2009.

[27] Wolf, T. *Replication of non-deterministic objects*. PhD thesis, Lausanne, 1998.

[28] Apache Hadoop. `http://hadoop.apache.org/`. Último acesso em 17 de agosto de 2010.

[29] Licenses- The Apache Software Foundation. `http://www.apache.org/licenses/`. Último acesso em 17 de agosto de 2010.

[30] Apple Xsan2. `http://images.apple.com/xsan/docs/L363053A_Xsan2_TO.pdf`. Último acesso em 17 de agosto de 2010.

[31] DRBD. `http://www.drbd.org`. Último acesso em 17 de agosto de 2010.

[32] Facebook has the world's largest Hadoop cluster! `http://hadoopblog.blogspot.com/2010/05/facebook-has-worlds-largest-hadoop.html`. Último acesso em 17 de agosto de 2010.

[33] Google facts and figures (massive infographic). `http://royal.pingdom.com/2010/02/24/google-facts-and-figures-massive-infographic/`. Último acesso em 17 de agosto de 2010.

[34] Hadoop Contributors Metting held on may 28th, 2010. `http://wiki.apache.org/hadoop/HadoopContributorsMeeting20100528`. Último acesso em 17 de agosto de 2010.

[35] HDFS UpRight overview. http://code.google.com/p/upright/wiki/HDFSUpRightOverview. Último acesso em 17 de agosto de 2010.

[36] Heartbeat. http://www.linux-ha.org/wiki/Heartbeat. Último acesso em 17 de agosto de 2010.

[37] Hunting disease origins with whole-genome sequencing. http://www.technologyreview.com/biomedicine/24720/. Último acesso em 17 de agosto de 2010.

[38] Lustre. http://www.lustre.org. Último acesso em 17 de agosto de 2010.

[39] Streaming edits to a standby name-node. http://issues.apache.org/jira/browse/HADOOP-4539. Último acesso em 17 de agosto de 2010.

[40] Worldwide LHC Computing Grid. http://public.web.cern.ch/public/en/lhc/Computing-en.html. Último acesso em 17 de agosto de 2010.

[41] Yahoo! Launches World's Largest Hadoop Production Application. http://developer.yahoo.net/blogs/hadoop/2008/02/yahoo-worlds-largest-production-hadoop.html. Último acesso em 15 de agosto de 2010.