Proposta de Dissertação de Mestrado

Estudo e Implementação de Mecanismos de Codificação por Apagamento no Hadoop *File System*

Celina d'Ávila Samogin
Instituto de Computação — UNICAMP

27 de agosto de 2010

Orientadora: Profa. Dra. Islene Calciolari Garcia

Resumo

Os dados em um sistema distribuído confiável devem estar disponíveis quando for necessário. A codificação por apagamento (*erasure codes*) tem sido utilizada por sistemas para alcançar requisitos de confiabilidade e de redução do custo de armazenamento de dados. O Hadoop é um *framework* para execução de aplicações em armazenamento distribuído de grande volume de dados e que pode ser construído com *commodity hardware*, que é facilmente acessível e disponível. Esta proposta apresentará uma análise da viabilidade da implementação prática de técnicas de codificação por apagamento no Hadoop *Distributed File System* (HDFS), as alterações no Hadoop e a eficácia dessas alterações. Esta proposta é uma contribuição para *software* livre em sistemas distribuídos.

1 Introdução

A codificação por apagamento (*erasures codes*) introduz redundância em um sistema de transimissão ou armazenamento de dados de maneira a permitir a detecção e correção de erros. A codificação por apagamento é, desde os anos 70, utilizada pela *NASA's Deep Space Network* para receber sinais e dados de telemetria (*downlinks*) vindos de veículos espaciais (*very distant spacecrafts*) e para enviar telecomandos (*uplinks*) para veículos espaciais [16, 21, 28].

A técnica de codificação por apagamento pode ser combinada com a distribuição de dados entre vários dispositivos de armazenamento, o que permite o aumento da largura de banda e a correção de erros [18, 25]. Requisitos de confiabilidade e de redução do tamanho do armazenamento podem ser observados em sistemas que tratam de: *digital fountain* (*multicasting* multimídia confiável)[1]; *Delay and Disruption Tolerant Networks*, redes de sensores e redes *peer-to-peer* [4, 3, 26] e armazenamento de grande volume de dados [2, 5, 12], como também o sistemas de arquivos distribuído do Hadoop (HDFS) [9].

O HDFS, por padrão, implementa alta disponibilidade dos dados via replicação simples dos blocos de dados. Esta abordagem acarreta um alto custo de armazenamento para garantir que os dados estarão sempre disponíveis. O objetivo do uso da codificação por apagamento no HDFS é permitir que o espaço de armazenamento possa ser reduzido sem prejudicar a disponibilidade dos dados. Esforços iniciais nessa linha foram feitos utilizando técnicas de RAID [9] e mais recentemente do algoritmo Reed-Solomon [11].

Este trabalho pretende avançar esta linha de pesquisa a partir dos seguintes passos:

- avaliação de desempenho, ganhos, e custos de diferentes estratégias de codificação por apagamento;
- implementação de otimizações ou extensões para o código que atualmente implementa Reed-Solomon, tentando melhorar, principalmente, a parte de distribuição de blocos;

- implementação de novos algoritmos (e.g., Tornado codes) e extenção da interface atual para aceitá-los;
- integração do código atual com o HDFS.

O texto a seguir está organizado da seguinte maneira: a Seção 2 introduz os conceitos básicos da codificação por apagamento, a Seção 3 comenta o *framework* Hadoop e seu sistema de arquivos, a Seção 4 apresenta os objetivos deste trabalho e a seção 5 cita as atividades propostas e o cronograma de execução.

2 Codificação por Apagamento

A codificação de mensagens no emissor antes da transmissão e a decodificação das mensagens (possivelmente danificadas) que chegam ao receptor, possibilita reparar os efeitos de um canal físico com ruídos [6] sem sobrecarregar a taxa de transmissão de informação ou o *overhead* de armazenamento [14].

Existem dois métodos básicos para tratar erros em comunicação e ambos envolvem a codificação de mensagens. A diferença está em como esses códigos são utilizados. Em um *repeat request system*, os códigos são utilizados para detectar erros e se estes existirem, é feito um pedido de retransmissão. Com *forward error correction*, os códigos são usados para detectar e corrigir erros.

Na Figura 1, vemos um sistema que utiliza código de blocos. A fonte envia uma sequência de dados para o codificador. O codificador divide esta sequência em m blocos de k *bits* cada chamados mensagens. Uma mensagem é representada por uma k-tupla binária $u = u_1, u_2, \ldots, u_k$. O codificador insere *bits* redundantes (ou de paridade) para cada mensagem u, gerando uma sequência de saída de n *bits* chamada *codeword* ou palavra código representada por uma n-tupla de símbolos discretos $v = v_1, v_2, \ldots, v_n$. Os $n - k$ bits são os bits redundantes que provêm à codificação a capacidade de tratar os ruídos do canal.

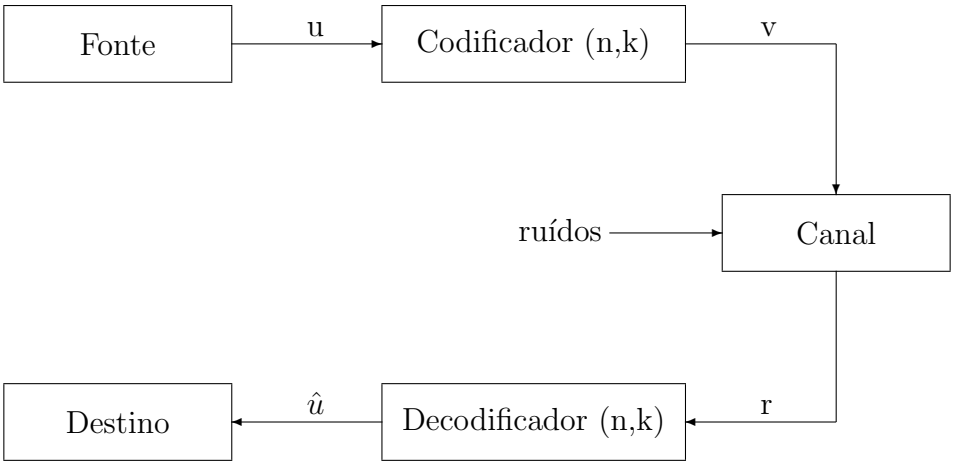

Figura 1: Códigos de bloco

A taxa de codificação e o *overhead* de armazenamento são calculados a partir de m blocos originais [3, 7]. São gerados n símbolos pelo algoritmo de codificação. $R = \frac{k}{n}$ é a taxa de codificação que pode ser interpretada como o número de bits de informação por palavra código transmitida e $O = \frac{1}{R}$ é o *overhead* de armazenamento.

Outra métrica utilizada é a redundância que pode ser definida por $\frac{(n-k)}{n}$. A alta redundância aumenta a possibilidade de todos os dados serem enviados em uma única transmissão. A desvantagem da redundância é que a adição de *bits* pode exigir uma largura de banda transmissão maior ou aumentar o atraso das mensagens (ou ambos).

Segundo [18], para sistemas de armazenamento, a codificação por apagamento baseada em operações simples, tais como XOR RAID, são preferíveis. Embora um mecanismo externo deva ser utilizado para detectar erros, as operações de XOR podem ser realizadas rapidamente e resultar em alto *throughput* das operações de codificação e decodificação.

Segundo [16], códigos Reed-Solomon (RS) são particularmente úteis para correção de erros em rajada (seqüência símbolos consecutivos, nenhum desses recebidos corretamente, chamados *burst errors*). Também podem ser usados eficientemente em canais em que o conjunto de símbolos de entrada é consideravelmente grande.

Códigos Tornado são uma classe de códigos LDPC (Low Density Parity Check) que utiliza grafos irregulares e que foi proposta por M. Luby [18]. Segundo [2], são mais

rápidos para codificar e decodificar e necessitam de um pouco mais de m fragmentos para reconstruir a informação. Em [1], o autor comentou o tempo de decodificação para códigos RS e Tornado. Códigos Tornado usam equações com um número pequeno de variáveis em contraste com códigos RS.

Na tabela 1, alguns sistemas que utilizam com codificação por apagamento foram comparados.

Replicação versus Codificação por Apagamento

Para implementar redundância de dados em sistemas, são utilizadas várias técnicas: codificação por apagamento, replicação, espelhamento, *Cyclic redundancy check* (CRC), *bits* de paridade, *checksum* e assinatura digital. Mecanismos de redundância podem implementar um conjunto destas técnicas [12].

A principal desvantagem da replicação é que ela requer um grande *overhead* de armazenamento para pouco ganho em disponibilidade e tolerância a falhas. Garantir que os dados permaneçam disponíveis quando todos os n dispositivos falham exige que, pelo menos, $n + 1$ cópias existam [18]. Em [20], os experimentos mostraram que o armazenamento no sistema Glacier utilizando replicação pura obteve um aumento de 11 vezes na quantidade de dados armazenados para conseguir 0.999999% de confiabilidade. Um cálculo da medida da quantidade de redundância da replicação pura e da codificação chamada *stretch factor* e a respectiva disponibilidade pode ser encontrado em [17].

Os autores em [22] afirmam que dados replicados permitem leituras de baixa latência, porque há muitas opções para a seleção de servidores, enquanto que dados codificados reduzem o consumo de largura de banda para escritas, em detrimento do aumento da latência de leituras.

Uma vantagem de codificação por apagamento seria um custo menor de armazenamento se comparado a replicação, no caso de grande volume de dados. Outra vantagem com relação a replicação foi comentada em [5]: para um mesmo espaço de armazenamento, o tempo médio entre falhas (*mean time to failure*) é maior.

Trabalho	Modelo	Objetivos	Codificação	Métricas
Byers,1998 [1]	*Digital Fountain*	Confiabilidade, Eficiência	Replicação, códigos Tornado e RS	tempo de codificação e decodificação de um bloco *bandwidth*, perda de pacotes
Weatherspoon, 2002 [5]	Arquivador central, nós	Disponibilidade	Replicação, Codificação por Apagamento	tempo médio entre falhas, *overhead* de armazenamento, tempo de verificação do bloco
Camargo, 2009 [8]	OppStore	Confiabilidade, Disponibilidade	Replicação	*bandwidth*, número de mensagens trocadas
Fan, 2009 [12]	HDFS	Confiabilidade, Disponibilidade	Replicação, RAID	atraso na codificação do bloco
Dabek, 2004 [22]	DHT	Eficiência	Replicação, Codificação por Apagamento	latência
Houri, 2009 [26]	*Peer-to-peer*	Disponibilidade	Replicação, Codificação por Apagamento	*bandwidth*
Plank, 2009 [30]	k discos de dados e m discos de paridade	Eficiência	códigos Tornado, RS e RAID	tempo de codificação de um grande arquivo de vídeo e tempo de decodificação de um *drive* de dados

Tabela 1: Comparação entre sistemas de codificação por apagamento

Em [2] e [5], os pesquisadores concluem que o uso de codificação por apagamento aumenta significativamente a disponibilidade de dados.

A Figura 2 apresenta um sistema de arquivos distribuídos armazenando um arquivo particionado em 8 blocos. O fator de replicação é 4. Os clientes precisam que, para cada um dos 8 distintos blocos, uma das quatro cópias esteja disponível. A Figura 3 apresenta o mesmo sistema que a Figura 2, mas utilizando códigos RS. O arquivo está particionado em 8 blocos e 32 blocos codificados foram gerados. Os clientes podem utilizar quaisquer 8 blocos para obter o arquivo inicial. A Figura 3 também se aplica a códigos Tornado.

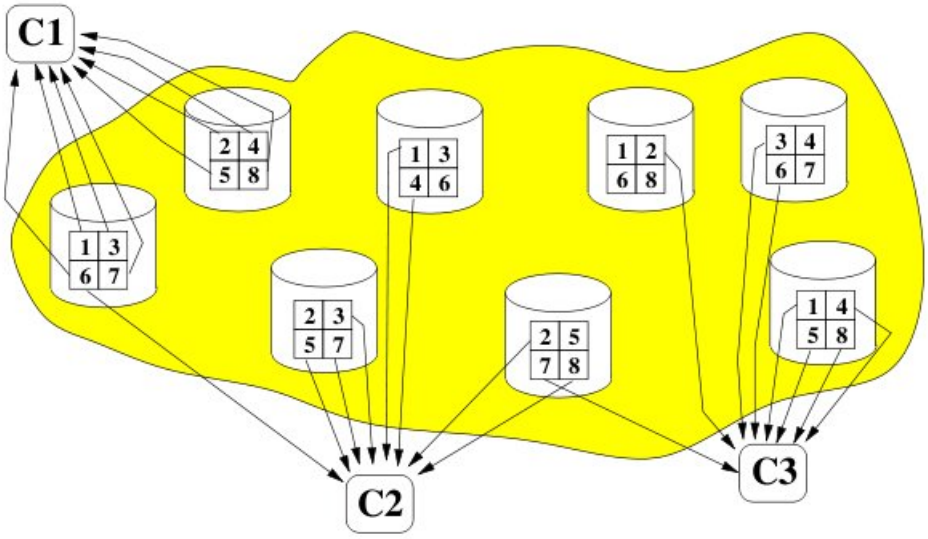

Figura 2: Sistema com replicação pura [13]

3 Hadoop

Atualmente, o Google é uma empresa de consulta e publicidade e é capaz de fornecer os seus serviços devido a investimentos em armazenamento distribuído em larga escala e a capacidade de processamento, estes desenvolvidos *in-house*.

Essa capacidade é fornecida por um grande número de PCs, pelo Google File System (GFS), um sistema de arquivos redundantes em *cluster*, pelo sistema operacional

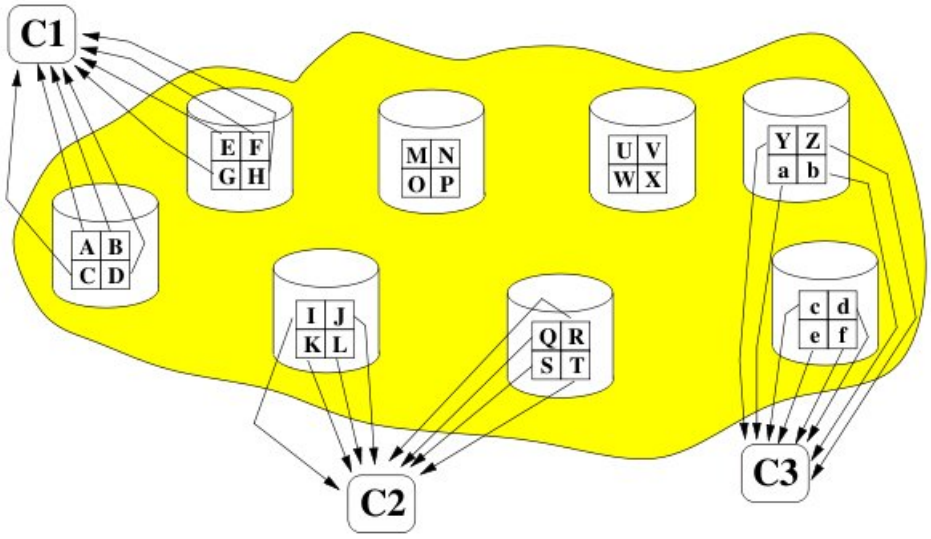

Figura 3: Sistema com códigos RS [13]

GNU/Linux e pelo MapReduce, um *middleware* de processamento paralelo de dados.

Em 2004, um artigo [24], que foi publicado por profissionais da Google, propôs o MapReduce. Em 2006, estes profissionais, juntamente com Doug Cutting do Yahoo!, formaram um sub-projeto do Apache Lucene[1] que foi chamado Hadoop[2].

Mais recentemente, o projeto Apache Hadoop tem desenvolvido uma reimplementação de partes do GFS e MapReduce e muitos grupos da comunidade de software livre posteriormente abraçaram essa tecnologia, permitindo-lhes fazer coisas que eles não poderiam fazer em máquinas individuais. O Hadoop está disponível em código fonte sob licenciamento Apache *license* (compatível com GPL).

O Hadoop é um *framework* para executar aplicações em armazenamento distribuído de grande volume de dados que pode ser construído com *commodity hardware*, que é facilmente acessível e disponível. O Hadoop não é um *framework* canônico. Ele foi projetado para aplicações que atualizam dados da seguinte forma: uma escrita e muitas leituras, através de acessos por *batch*, com tamanho da ordem de petabytes, organizados

[1]http://www.apache.org
[2]http://hadoop.apache.org/

de forma não estruturada, com esquema dinâmico e integridade baixa. Uma lista de aplicações e organizações que usam o Hadoop pode ser encontrada em [29].

Em poucas palavras, o Hadoop disponibiliza um armazenamento compartilhado (HDFS) e um sistema de análise (MapReduce) que compõem o seu *kernel*.

3.1 MapReduce

O MapReduce utiliza algoritmos de ordenação para reconstruir sua base de dados. Um bom uso para o MapReduce são aplicações cujos dados são escritos uma vez e lidos muitas vezes. São dados não estruturados como texto ou imagens. O MapReduce tenta colocar esses dados no nó onde são feitas as computações, desta forma, o acesso aos dados é rápido, pois é local [27].

O MapReduce pode resolver problemas genéricos, cujos dados podem ser divididos em matrizes de dados, para cada matriz a mesma computação necessária (sub-problema) e não existe necessidade de comunicação entre as tarefas (sub-problemas). A execução de um típico *job* do MapReduce pode ser assim descrita:

- Iteração sobre um número grande de registros
- Map extrai algo de cada registro (chave, valor)
- Rearranjo (*shuffle*) e ordenação de resultados intermediários por (chave, valor)
- Reduce agrega os resultados intermediários
- Geração da saída

Um programas para execução no HFS/MapReduce que podem ser escritos em várias linguagens como Java, Ruby, Python e C++.

3.2 Arquitetura do Hadoop *Distributed File System*

Um *cluster* do HDFS é composto por um único NameNode, um servidor-mestre que gerencia o sistema de arquivos e controla o acesso aos arquivos de clientes. Há uma

série de DataNodes, geralmente um por nó do *cluster*, que gerenciam o armazenamento anexado ao nó em que são executados. A Figura 5 mostra o NameNode e os DataNodes.

Uma típica arquitetura de rede em dois níveis para um *cluster* Hadoop é construída por vários *racks* interligados por um comutador como mostra a Figura 4. Cada *rack* por sua vez é formado por vários nós (máquinas) e seus discos, estes também interligados por um comutador.

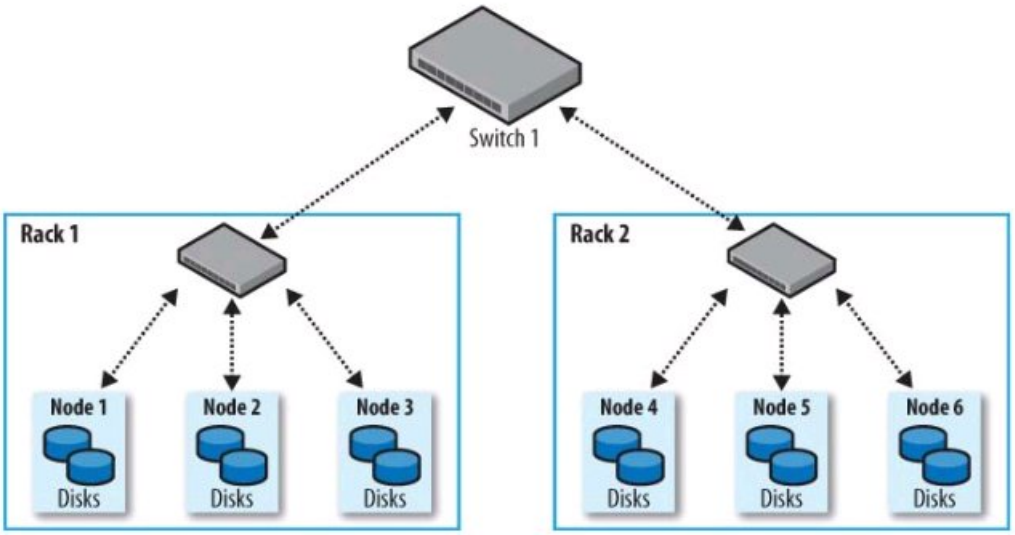

Figura 4: Arquitetura de rede em dois níveis para um cluster Hadoop [10]

O NameNode executa operações no sistema de arquivos, como *open*, *close*, *rename* de arquivos e de diretórios.

HDFS disponibiliza espaço para sistema de arquivos e permite que os dados do usuário sejam armazenados em arquivos. Internamente, um arquivo é dividido em um ou mais blocos e esses blocos são armazenados em um conjunto de DataNodes. A Figura 6 mostra DataNodes e seus blocos. O tamanho *default* de cada bloco é 64MB.

Os DataNodes respondem aos pedidos de leitura e escrita de clientes do sistema de arquivos e também executam a criação, eliminação e replicação de blocos sob instrução do NameNode. O número de réplicas é geralmente 3. A 1^a réplica fica local, no mesmo nó do código do cliente. A 2^a réplica fica em um nó em outro *rack* e a 3^a réplica fica

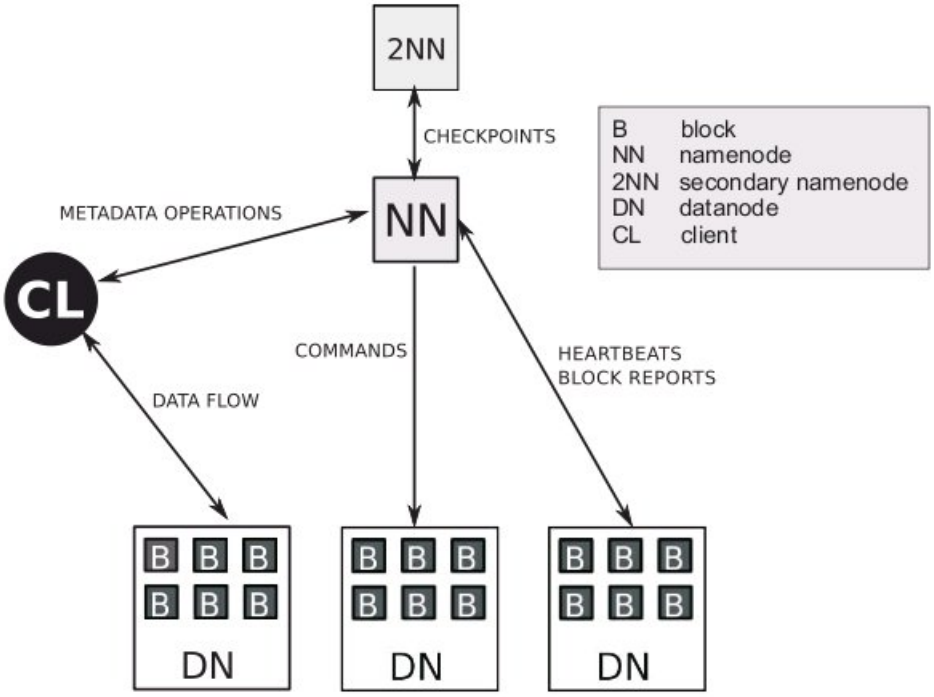

Figura 5: Arquitetura do HDFS [23]

nesse último *rack* em outro nó. As 2^a e 3^a réplicas não são locais ao bloco replicado.

O NameNode e DataNode são partes do *software* projetado para rodar em *commodity hardware*. Essas máquinas normalmente executam um sistema operacional GNU/Linux.

HDFS é construído usando a linguagem Java. Qualquer máquina que suporte Java pode executar o NameNode ou o DataNode [10].

Os protocolos do HDFS usam o protocolo TCP/IP. O cliente fala o protocolo ClientProtocol com o NameNode através de uma porta. Os DataNodes falam o protocolo DataNodeProtocol com o NameNode. Esses protocolos executam uma *Remote Procedure Call* (RPC). O NameNode não inicia chamadas RPCs. Ele responde a chamadas RPCs feitas pelo DataNodes e pelos clientes.

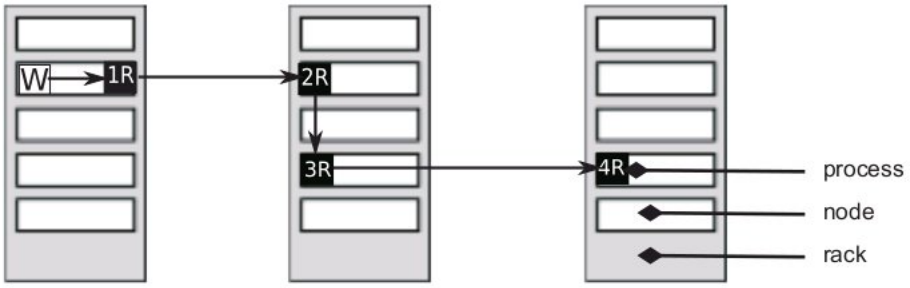

Figura 6: Arquitetura do HFS - Datanodes e Blocos [27]

3.3 Codificação por apagamento

Existe uma nova característica proposta em 2009 para implementação de uma camada de codificação por apagamento no Hadoop utilizando RAID [9] e uma mais recente utilizando códigos RS [11].

A versão atual do Hadoop utiliza apenas a técnica de replicação [27] para obter disponibilidade e confiabilidade de dados. A inclusão da codificação por apagamento será feita com o objetivo de reduzir o tamanho do armazenamento do HDFS.

4 Proposta

4.1 Objetivos

Esta proposta apresentará uma análise da viabilidade da implementação prática de várias técnicas de codificação por apagamento no HDFS, as alterações no Hadoop e a eficácia dessas alterações. Esta proposta é uma contribuição para *software* livre em sistemas distribuídos.

O objetivo do uso da codificação é reduzir o tamanho do armazenamento utilizando para isso a redução do fator de replicação de um bloco e codificação de um conjunto de blocos (a partir de um bloco inicial) de tal modo que a probabilidade de falha de um

bloco permaneça a mesma que antes ou diminua.

Os testes serão feitos para mostrar a eficácia das alterações no HDFS: quanto de espaço em disco foi economizado e o tempo de latência de leitura de arquivos. Também está prevista a implementação e validação em software dos algoritmos RS e Tornado, que possibilitará a validação da funcionalidade desses algoritmos.

4.2 Métodos

Este trabalho irá estender e alterar código fonte distribuído sob a licença Apache. Espera-se a interação e colaboração com os desenvolvedores. Atualmente, Rodrigo Malta Schimdt, ex-aluno do Instituto de Computação da Unicamp e um dos responsáveis pela inclusão de técnicas de codificação por apagamento no HDFS tem contribuído com várias ideias para a realização deste trabalho.

Os testes poderão utilizar:

- máquinas do Instituto de Computação da Unicamp, principalmente do LSD (Laboratório de Sistemas Distribuídos);
- máquinas do ambiente computacional do CENAPAD-SP (Centro Nacional de Processamento de Alto Desempenho em São Paulo)
- a nuvem do AWS (Amazon *Web Services*).

Na tabela 2 o Modelo 1 é modelo atual do HDFS, o Modelo 2 é modelo em implementação com característica HDFS-503 [9] e as Propostas 3, 4 e 5 são propostas de implementação de algoritmos de codificação para este trabalho. A disponibilidade, a probabilidade de corrupção de um arquivo e o espaço de armazenamento foram adaptadas de [11] e de [17]. A possibilidade de falhas nas máquinas estão atreladas a uma boa escolha da distribuição dos blocos pelos dispositivos de armazenamento.

Proposta/ Modelo	Codificação	Disponibilidade	Probabilidade de corrupção de um arquivo	Espaço de armazenamento
1	sem codificação, fator replicação = n	baixa em relação ao espaço de armazenamento	$O(p^n)$	nx
2	Códigos RAID, 1 bloco de paridade, stripe = 10 blocos, fator replicação = 2	permite falha em 1 máquina	$O(p^4)$	$2.2x$
3	Códigos RS RAID, 4 blocos de paridade, stripe = 10 blocos, fator replicação = 1	permite falha em 3 máquinas	$O(p^5)$	$1.4x$
4	Códigos RS, 4 blocos de paridade, fator replicação = 1, com n máquinas	permite falha em até $(3m + 1)$ máquinas	$\Omega(p^{3m+2})$	até $5x$
5	Códigos Tornado, 4 blocos de paridade, fator replicação = 1, com n máquinas	permite falha em até $(3m + 1)$ máquinas	$\Omega(p^{3m+2})$	até $5x$

Tabela 2: Comparação entre algoritmos de codificação por apagamento e replicação

em que:
p = probabilidade de perda do bloco, $0 < p < 1$
x = tamanho do armazenamento em disco de um bloco
m = número de fragmentos do bloco inicial antes da codificação
$n = 4m$ é número de blocos codificados a partir a um bloco inicial; o bloco codificado b_i está armazenado na máquina d_i, para $1 \leq i \leq m$; o bloco inicial e a primeira réplica estão na máquina $d1$

4.3 Forma de Análise dos Resultados

Nós poderemos utilizar alguns *ebooks* do Projeto Gutenberg e do Portal Domínio Público como entrada de dados de alguns dos testes:

- Teste de funcionalidade dos algoritmos de codificação e de decodificação RS e de outra codificação como Tornado
- Teste *cluster* Hadoop 0.21.0 que utiliza apenas replicação
- Teste *cluster* Hadoop 0.21.0 com característica HDFS-503 que utiliza replicação e codificação por apagamento [9]

Estamos prevendo duas fases de teste:

testes de funcionalidade e de injeção de falhas testar os algoritmos que criam os blocos codificados (dados e paridade) e os mantêm; testar os algoritmos que atendem os pedidos de leitura (réplica); testar os algoritmos que percebem réplicas indisponíveis e as reconstroem a partir dos blocos codificados (para isso utilizar possivelmente o Zookeeper, um serviço de coordenação de processos de aplicações em sistemas distribuídos); testar os algoritmos que percebem blocos indisponíveis e reconstroem as réplicas (se indisponíveis); esta fase será executada em ambiente virtualizado;

testes de desempenho, de tamanho do armazenamento e de injeção de falhas obter uma aproximação do tamanho do armazenamento (dados e paridade) para conjuntos de arquivos que ocupem espaço original do tamanho de alguns gigabytes e terabytes; esta fase será executada em ambiente o mais real possível.

Os algoritmos de codificação e de decodificação poderão permitir parametrizar:

- número de pedaços que o bloco original é dividido antes da geração dos blocos codificados

- número de blocos de paridade (redundância)
- fator de replicação

4.4 Plano de Trabalho e Cronograma

O mestrado iniciou-se em março de 2010 e o cronograma estende-se até março de 2012. As fases do cronograma são:

1. Créditos do mestrado
2. Exame de qualificação do mestrado
3. Revisão bibliográfica
4. Implementação
5. Realização de testes
6. Escrita da dissertação de mestrado
7. Preparação de artigo para congresso ou revista
8. Defesa da dissertação

	2010					2011						2012
Atividade	3–4	5–6	7–8	9–10	11–12	1–2	3–4	5–6	7–8	9–10	11–12	1–2
1	o	o	o	o	o							
2				o								
3	o	oo	oo	oo	oo	oo	oo	o	o	o	o	
4			o	oo	oo	oo	oo	oo	oo	oo	oo	
5		o	o	oo	oo	oo	oo	oo	oo	oo	oo	
6		oo	oo	oo	oo	oo	o	o	o	o	o	
7								o	o	o		
8												o

4.5 Contribuições deste trabalho

Overview de Codificação por Apagamento

A revisão bibliográfica desse tema tem exigido tempo e dedicação, devido a existência de poucas pesquisas experimentais publicadas sobre o tema. Uma dificuldade encontrada por quem estuda codificação por apagamento é que não existe uma nomenclatura unificada [30]. Também segundo [15], existem poucos pesquisadores que são programadores de sistemas e que fazem propostas neste tema.

Uma classificação para códigos de blocos pode ser encontrada em [19].

4.5.1 Submeter as alterações e sugestões como contribuição ao Hadoop

As alterações e sugestões para uso das codificações por apagamento no Hadoop serão propostas a comunidade por meio do site da Apache *Software Foundation*, como foi proposta a versão inicial de codificação por apagamento no HDFS [9] e a segunda versão com códigos Reed-Solomon [11].

Referências

[1] Byers, J. W. Luby, M. Mitzenmacher, M. Rege, A. A digital fountain approach to reliable distribution of bulk data. *SIGCOMM Computer Communication Rev.*, 28(4):56–67, 1998.

[2] Kubiatowicz, J. Bindel, D. Chen, Y. Czerwinski, S. Eaton, P. Geels, D. Gummadi, R. Rhea, S. Weatherspoon, H. Weimer, W. Wells, C. Zhao, B. Oceanstore: an architecture for global-scale persistent storage. *SIGPLAN Not.*, 35(11):190–201, 2000.

[3] Oliveira, C. T. Moreira, M. D. D. Rubinstein, M. G. Costa, L. H. M. K. Duarte, O. C. M. B. Mc05: Redes tolerantes a atrasos e desconexões. In *Anais do 25o Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos*, Belém, Pará, Brasil, May 2007.

[4] Rodrigues, R. Liskov, B. High availability in dhts: Erasure coding vs. replication. In *Peer-to-Peer Systems IV*, pages 226–239. LNCS, 2005.

[5] Weatherspoon, H. Kubiatowicz, H. J. D. Erasure coding vs. replication: A quantitative comparison. In *IPTPS '01: Revised Papers from the First International Workshop on Peer-to-Peer Systems*, pages 328–338, London, UK, 2002. Springer-Verlag.

[6] Shannon, C. E. A mathematical theory of communication. *The Bell System Technical Journal*, 27(3):379–423, 623–656, 1948.

[7] Alan Sussman et al. lectures of peer-to-peer and grid computing course. URL=http://www.cs.umd.edu/class/spring2007/cmsc818s/Lectures/lectures.htm. Acessado em 03 de maio de 2010.

[8] Camargo, R. Y. Filho, F. C. Kon, F. Efficient maintenance of distributed data in highly dynamic opportunistic grids. In *SAC '09: Proceedings of the 2009 ACM Symposium on Applied Computing*, pages 1067–1071, New York, NY, USA, 2009. ACM.

[9] Apache Software Foundation. Hdfs-503 - implement erasure coding as a layer on hdfs. URL=https://issues.apache.org/jira/browse/HDFS-503. Acessado em 08 de maio de 2010.

[10] Apache Software Foundation. Hdfs architecture. URL=http://hadoop.apache.org/common/docs/current/hdfs_design.html. Acessado em 08 de maio de 2010.

[11] Apache Software Foundation. Mapreduce-1969 - allow raid to use reed-solomon erasure codes. URL=https://issues.apache.org/jira/browse/MAPREDUCE-1969. Acessado em 20 de agosto de 2010.

[12] Fan, B. Tantisiriroj, W. Xiao, L. Gibson, G. Diskreduce: Raid for data-intensive scalable computing. In *PDSW '09: Proceedings of the 4th Annual Workshop on Petascale Data Storage*, pages 6–10, New York, NY, USA, 2009. ACM.

[13] Plank, J. S. Thomason, M. G. A practical analysis of low-density parity-check erasure codes for wide-area storage applications. In *DSN*, pages 115–124, 2004.

[14] Lin, S. Costello, D. J. J. *Error Control Coding: Fundamentals and Applications*, chapter 1, pages 1–14. Prentice-Hall Press, Englewood Cliffs, New Jersey, USA, 1983.

[15] Plank, J. Cs560 - operating systems. URL=http://www.cs.utk.edu/plank/plank/classes/cs560. Acessado em 08 de maio de 2010.

[16] Almeida, G. M. Códigos corretores de erros em hardware para sistemas de telecomando e telemetria em aplicações espaciais. Master's thesis, Pontifícia Universidade Católica do Rio Grande do Sul - Faculdade de Informática, Porto Alegre, Brasil, março 2007.

[17] Bhagwan, R. Moore, D. Savage, S. Voelker, G. M. Replication strategies for highly available peer-to-peer storage. In *Future directions in distributed computing: research and position papers*, pages 153–158, Berlin, Heidelberg, 2003. Springer-Verlag.

[18] Woitaszek, M. *Tornado Codes for Archival Storage*. PhD thesis, Department of Computer Science of the University of Colorado, Boulder, Colorado, USA, dezembro 2007.

[19] MathWorks. Error detection and correction. URL=http://www.mathworks.com/access/helpdesk/help/toolbox/comm/ug/fp6603.html. Acessado em 04 de julho de 2010.

[20] Haeberlen, A. Mislove, A. Druschel, P. Glacier: highly durable, decentralized storage despite massive correlated failures. In *NSDI'05: Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation*, pages 143–158, Berkeley, CA, USA, 2005. USENIX Association.

[21] Curtis, A. R. Space today online - communicating with interplanetary spacecraft. URL=http://www.spacetoday.org/SolSys/DeepSpaceNetwork/DeepSpaceNetwork.html. Acessado em 03 de maio de 2010.

[22] Dabek, F. Li, J. Sit, E. Robertson, J. Kaashoek, M. F. Morris, R. Designing a dht for low latency and high throughput. In *NSDI'04: Proceedings of the 1st Conference on Symposium on Networked Systems Design and Implementation*, pages 85–98, Berkeley, CA, USA, 2004. USENIX Association.

[23] Oriani, A. Garcia, I. C. Schmidt, R. The Search for a Highly-Available Hadoop Distributed Filesystem. Technical Report IC-10-24, Institute of Computing, University of Campinas, August 2010.

[24] Dean, J. Ghemawat, S. Mapreduce: simplified data processing on large clusters. In *OSDI'04: Proceedings of the 6th conference on Symposium on Opearting Systems Design & Implementation*, pages 137–150, Berkeley, CA, USA, 2004. USENIX Association.

[25] Plank, J. S. A tutorial on reed-solomon coding for fault-tolerance in raid-like systems. *Software Practice Experience*, 27(9):995–1012, 1997.

[26] Houri, Y. Jobmann, M. Fuhrmann, T. Self-organized data redundancy management for peer-to-peer storage systems. In *IWSOS '09: Proceedings of the 4th IFIP TC 6 International Workshop on Self-Organizing Systems*, pages 65–76, Berlin, Heidelberg, 2009. Springer-Verlag.

[27] White, T. *Hadoop: The Definive Guide*, chapter 1, 2, 3, pages 1–74. O'Reilly, Sebastopol, CA, USA, 2009.

[28] Sniffin, R. W. Telemetry data decoding. URL=http://deepspace.jpl.nasa.gov/dsndocs/810-005/208/208A.pdf. Acessado em 03 de maio de 2010.

[29] Hadoop Wiki. Hadoop wiki - poweredby. URL=http://wiki.apache.org/hadoop/PoweredBy. Acessado em 08 de maio de 2010.

[30] Plank, J. S. Xu, L. Luo, J. Schuman, C. D. Wilcox-O'Hearn, Z. A performance evaluation and examination of open-source erasure coding libraries for storage. In *FAST '09: Proceedings of the 7th Conference on File and Storage Technologies*, pages 253–265, Berkeley, CA, USA, 2009. USENIX Association.