Instituto de Computação
Universidade Estadual de Campinas

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

Celina d'Ávila Samogin

Este exemplar corresponde à redação da Dissertação apresentada para a Banca Examinadora antes da defesa da Dissertação.

Campinas, 05 de Março de 2012.

Profa. Dra. Islene Calciolari Garcia
(Orientadora)

Dissertação apresentada ao Instituto de Computação, UNICAMP, como requisito parcial para a obtenção do título de Mestre em Ciência da Computação.

Instituto de Computação
Universidade Estadual de Campinas

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

Celina d'Ávila Samogin[1]

30 de Janeiro de 2012

Banca Examinadora:

- Profa. Dra. Islene Calciolari Garcia (Orientadora)
- Prof. Ph.D. Luiz Eduardo Buzato
- Prof. Ph.D. Ricardo Dahab
- Prof. Dr. Edmundo Roberto Mauro Madeira (suplente)

[1]Suporte financeiro de: Bolsa do CNPq (processo XYZ) 2010–2010.

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. Este trabalho apresenta os aspectos teóricos envolvidos e uma implementação prática de técnicas de codificação por apagamento no Hadoop *Distributed File System* (HDFS), as alterações no Hadoop e a eficácia dessas alterações. Este trabalho é uma contribuição para *software* livre em sistemas distribuídos.

Abstract

The data in a reliable distributed system should be available when needed. Erasure codes have been used by systems to meet reliability requirements and reduce the cost of data storage. The Hadoop is a framework for running applications on distributed storage of large volumes of data and it can be built with commodity hardware, which is easily accessible and available. This dissertation presents theorical aspects involved and a practical implementation of erasure coding techniques in Hadoop Distributed File System (HDFS), changes in Hadoop and effectiveness of those changes. This work is a contribution to free software in distributed systems.

Agradecimentos

Agradeço minhas irmãs Eunice e Helena, que sempre me apoiaram nas minhas conquistas (foram muitas) e nos meus fracassos (foram poucos) e meus filhos Paula, Carolina e Daniel que entenderam minha ausência. Meus tios e tias, meus primos e primas também sempre me apoiando, querendo saber das novidades e perguntando: "quando você vem em Lins ?". A família é mesmo minha fortaleza.

Meus queridos amigos de quatro patas também tiveram muito a ver com esse trabalho. Uma parte da viabilização do trabalho foi obtida do faturamento das aulas de adestramento para cães e gatos que eu fiz entre 2005 e 2009. Depois disso, eu me afastei do trabalho com proprietários e continuei com o trabalho voluntário para as ONGs de posse responsável e castração, continuando a treinar animais resgatados das ruas para que eles ficassem mais adotáveis. A profa Maria de Fátima Martins (USP-FMVZ) e suas alunas Msc. Michele Ribeiro Silva e Msc. Juliana de Vazzi Pinheiro também foram muito importantes nesse processo difícil de voltar a academia. Ela me aconselhava em conversas: "Ela não tem que fazer outra faculdade!".

O colega de turma e amigo Paulo Cezar Campioni também apoiou meu trabalho de mestrado. Foi um pai sempre presente com nossos filhos.

Agradeço ao CNPq (Conselho Nacional de Desenvolvimento Científico e Tecnológico) pelo apoio concedido no início do meu mestrado e a FAPESP (Fundação de Amparo à Pesquisa do Estado de São Paulo) pelo apoio concedido ao meu projeto "Desenvolvimento e Qualidade de Dados para Sistema de Informação de Biodiversidade (Projeto de Pesquisa: Desenvolvimento de novas ferramentas e migração dos sistemas BIOTA do CRIA para o CENAPAD/UNICAMP)".

Agradeço aos professores, membros da banca. É uma honra contar com suas contribuições para meu trabalho.

O meu agradecimento também vai para os coordenadores de graduação e pós-graduação do IC, nesse período de 2010 a 2011, profo Ricardo da Silva Torres, profo Sandro Rigo, profo Hélio Pedrini, profo Alexandre Xavier Falcão e profo Paulo Lício de Geus.

Eu não queria falar muito do passado, mas eu tentei iniciar no programa de pós-graduação, logo que eu terminei a graduação em 1985 e uma outra vez, em 2002. Eu

acredito que viram meu potencial, pois me aceitaram pela terceira vez no IC em 2010 e eu pude, então, concluir o programa de mestrado.

Também gostaria de agradecer minha colega e amiga Claudia Marconi Engler Pinto, da turma de ciência da computação de 1981, pela palestra sobre Unix, ocorrida um tempo depois da minha formatura na graduação, que me despertou o interesse por esse sistema operacional. Ao CPqD Telebrás, hoje, Fundação CPqD, por ter me oferecido a oportunidade de assistir a palestra apresentada pela colega Claudia e de cursar disciplinas do Laboratório A-HAND, na época com o coordenador, o profo Rogério Drummond, onde eu pude aprender, com muita qualidade, a programar em sistemas Unix, muito antes de eu conhecer o sistema Linux e os sistemas MS Windows.

Já algum tempo, antes de 2008, quando eu quis aprender a teoria de computação que estava sendo oferecida nos cursos de graduação do IC, as páginas dos cursos do profo Rezende foram (e continuam a ser) importantes fontes de informação sobre essa área e me ajudaram muito a direcionar meus estudos. Obrigada, professor Rezende, por me aceitar como ouvinte nas disciplinas de Fundamentos Matemáticos da Computação (MC348) e Projeto e Análise de Algoritmos I (MC448), por proporcionar aulas tão bem elaboradas, lousas perfeitas e por demonstrar sempre disposição com as dúvidas que você me respondeu nos horários de atendimento. Fazer os quase 500 exercícios propostos nessas aulas me prepararam para as disciplinas da pós-graduação.

Também não posso me esquecer dos colegas de ciência da computação de 2007 que me adotaram como parte da turma, afinal, além das disciplinas com o profo Rezende, eu assisti Banco de Dados (MC536) com eles também! As animadas e surpreendentes aulas da profa Claudia também me prepararam para a pós. Obrigada Alex Grilo, Michael, Vanessa Schissato, Douglas Drummond, Bruno Malveira, Guilherme Paulovic, Guilherme Sampaio, Helen Her, Bráulio, "Panda"e demais colegas, que eu lembro a fisionomia, mas não anotei o nome.

Agradeço também o profo Meidanis por me fazer estudar todos os dias nos meses de aulas de Complexidade de Algoritmos I (MO417). As técnicas utilizadas e o formato das aulas e das avaliações exigiram de mim mais participação na disciplina e diminuíram meu estresse com relação aos tópicos da disciplina. Foi uma boa convivência com meus colegas dessa turma também. Não teve turma mais legal!

Entre outras coisas, a profa Ariadne e o profo Buzato me ensinaram como registrar e escrever resumos de artigos, que "comunicam em uma página a tese apresentada pelos autores e a sua relevância", o que é essencial para o desenvolvimento do trabalho da pós-graduação. Agradeço muito a eles por isso.

A oferta da disciplina Projeto e Implementação de Sistemas Distribuídos (MO641) pelo profo Buzato com trabalhos e seminários focados no projeto do Hadoop, também foi muito importante, pois pude conhecer e implementar aplicações que esse sistema distribuído de

arquivos pode suportar.

Com o apoio da docente, a Profa Islene, minha orientadora, eu tive uma boa e interessante experiência para minha carreira de docente no estágio PED da disciplina Laboratório de Sistemas Distribuídos (MC715). Aprendi muito assistindo as apresentações dos alunos, inclusive como capturar imagem do monitor para criar um vídeo e instalando, compilando e testando o código java dos projetos dos alunos no Cluster do IC!

Obrigada Profa Islene, que sempre acreditou no meu potencial e me conduziu durante o mestrado. Pela sua atenção, paciência, experiência, tranquilidade, sorrisos, incentivo e críticas que contribuíram para o meu crescimento como aluna e como pessoa. Foi um grande privilégio tê-la como orientadora.

Não poderia deixar de agradecer o profo Tomasz pelas ótimas aulas que assisti na disciplina Estrutura de Dados (MC202) no programa de estágio PED e por ele sempre mostrar tanta qualidade e entusiasmo como professor. Responder as dúvidas no laboratório e por e-mail e fazer uma avaliação das tarefas dos alunos da turma de ciência da computação de 2011, com ajuda do colega Thiago Cavalcante, me estimularam a desenvolver técnicas para meu aprimoramento como docente.

Agradeço muito ao Rodrigo Schmidt do Facebook, ex-aluno do IC, pelas sugestões de temas, das quais escolhi *erasure coding* e pela sua disposição em responder minhas dúvidas sobre a camada RAID do Hadoop. Não foi só ele que me ajudou nessa tarefa de conhecer o Hadoop. Também agradeço Scott Chen e Ramkumar Vadali.

Com a muito competente equipe da secretaria de pós-graduação, Daniel Capeleto, Fernando Okabe, Wilson Bagni Junior e Ademilson Ramos dos Reis, meu período como aluna no IC foi muito tranquilo. Pude contar com eles para todo apoio e para resolver qualquer problema.

Também quando um apoio de informática foi importante, Wiliam Lima Reiznautt e sua equipe sempre estavam dispostos a resolver a questão.

Sem música, alguns dias teriam sido muito difícies. Obrigada Paralamas do Sucesso, Skank, Legião Urbana, Capital Inicial, Tim Maia, Erasmo Carlos, Mutantes, Rita Lee, Celly Campello, Raul Seixas, Elis Regina, Nara Leão, João Gilberto, Chico Buarque, Paulinho da Viola, Demônios da Garoa, Carlos Lyra, Tom Jobim, Vinicius de Moraes, Tiê, Trio Irakitan, Almir Sater, Cascatinha e Inhana, Fafá de Belém, Marisa Monte, Arnaldo Antunes, Adriana Calcanhoto, Renato Russo, Pepeu Gomes, Caetano Veloso, Gilberto Gil, Léo Jaime, Cazuza, Metrô, Premeditando o Breque, Lobão, Camisa de Vênus, Ira!, Roupa Nova, 14 Bis, Blitz, Kid Abelha, Morris Albert, Norah Jones, Aretha Franklin, B. J. Thomas, Nat King Cole, Elvis Presley, Chuck Berry, Jerry Lee Lewis, Roy Orbison, Frank Sinatra, The Platters, Bob Dylan, Al Stewart, Cat Stevens, The Beatles, Suzi Quatro, Rolling Stones, Supertramp, Aerosmith, Nirvana, Led Zeppelin, Talking Heads, Sixpence None the Richer, Five, Dire Straits, Aqualung, The Police, The Who,

REM, Duran Duran, Peter Frampton, Art Garfunkel, John Lennon, Paul McCartney, George Harrison, Ringo Star, Paul Simon, David Bowie, Carlos Santana, Eric Clapton, Elton John, Carly Simon, Jack Johnson, Lynyrd Skynyrd, Beach Boys, Player, Three Dog Night, The Eagles, The Doors, Amy Winehouse, Bread, Carpenters, The Mamas and The Papas, Joan Baez, Lobo, The Monkees, Pink Floyd, The Alan Parsons Project, Nicolette Larson, Kate Bush, Laura Pausini, Dolly Parton, Earth, Wind and Fire, Michael Jackson e Avril Lavigne pela voz, letras e melodias e muitos outros artistas pelas músicas com arranjos para violão, gaita de fole e saxofone e com solos de guitarra.

Agradeço ainda toda comunidade de software livre por todas as informações que pude encontrar nos fóruns, nas listas de discussão, nos sites, nas documentações, por toda a conquista do software de qualidade obtida com o trabalho colaborativo, software esse testado por milhares de pessoas e em centenas de projetos e empresas. É nisso que eu acredito!

Dedico ao meu pai Antônio (in memoriam), à minha mãe Maria (in memoriam), às pessoas que, de alguma maneira, com suas atitudes, contribuíram para a realização deste trabalho e a todos que dedicam sua vida a pesquisa científica.

Há homens que lutam um dia, e são bons; há outros que lutam um ano, e são melhores; há aqueles que lutam muitos anos, e são muito bons. Porém há os que lutam toda a vida, esses são os imprescindíveis. (Bertolt Brecht, "Os Que Lutam")

Sumário

Lista de Tabelas

Lista de Figuras

Lista de Abreviaturas

bit BInary digiT

BCH Bose, Chaudhuri and Hocquenghem

BSC Binary Symmetric Channel

BSD Berkeley Software Distribution

CD Compact Disk

CCSDS Consultative Committee for Space Data Systems

CRC Cyclic Redundancy Check

DSN Deep Space Network

DVD Digital Versatile Disc

ECC Error Correcting Code

GF Galois Field

GPL GNU General Public License

GPL2 GPL versão 2

IEEE Institute of Electrical and Electronics Engineers

LFSR Linear Feedback Shift Register

LGPL GNU Lesser General Public License

MAID Massive Arrays of Idle Disks

MDS Maximum Distance Separable

NASA National Aeronautics and Space Administration

PNG Portable Network Graphics

POSIX Portable Operating System Interface

RAID Redundant Arrays of Independent Disks

RS Reed-Solomon

WLAN Wireless Local Area Network

WMAN Wireless Metropolitan Area Network

WiMAX Worldwide Interoperability for Microwave Access

Wi-Fi marca da Wi-Fi Alliance

XOR Exclusive OR

Capítulo 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 [38, 49, 66].

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 [42, 56]. Requisitos de confiabilidade e de redução do tamanho do armazenamento podem ser observados em sistemas que tratam de: *Delay and Disruption Tolerant Networks*, redes de sensores e redes *peer-to-peer* [39, 45, 6, 5, 7, 59] e armazenamento de grande volume de dados [10, 4, 53, 57, 28, 35, 36], como também o sistema de arquivos distribuído do Hadoop (HDFS) [18].

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 [18] e mais recentemente do algoritmo Reed-Solomon [20].

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: os Capítulos 2 e 3 introduzem os conceitos básicos da álgebra abstrata e da codificação por apagamento, respectivamente, o Capítulo 4 apresenta uma discussão sobre esquemas de redundância de dados, o Capítulo 5 comenta o *framework* Hadoop e seu sistema de arquivos, o Capítulo 6 comenta as codificações Tornado e Turbo-*Like* que foram implementadas para uma versão do Hadoop e o Capítulo 7 apresenta as contribuições e conclusões deste trabalho.

Capítulo 2

Álgebra Abstrata

In Galois fields, full of flowers.
Primitive elements dance for hours...
Stephen B. Weinstein

Esse capítulo tem por objetivo apresentar conceitos matemáticos fundamentais dentro do escopo de álgebra abstrata para entendimento de Códigos de Blocos [58].

Códigos BCH e RS são projetados através da aritmética de corpos finitos. Corpos finitos também são chamados corpos de Galois, em homenagem ao matemático francês Évariste Galois [25].

2.1 Definições em $\mathbb{Z}$

Os alfabetos que utilizamos nas definições deste capítulo são o conjunto $\mathbb{Z}$ e o seu subconjunto $\mathbb{A} = \{0, 1\}$.

Definição 2.1 Congruência Linear *Seja $q \in \mathbb{Z}$. Dois inteiros a e b dizem-se congruentes módulo$-q$ se tiverem o mesmo resto na divisão por q. A notação é $a \equiv b(mod\ q)$. Daí temos que $a = qk + b$, para um $k \in \mathbb{Z}$.*

Exemplo 2.1

$$32 \equiv 2(mod\ 3)$$
$$27 \equiv 5(mod\ 11)$$
$$63 \equiv 7(mod\ 8)$$

Definição 2.2 Classe Residual *Seja $q \in \mathbb{Z}$ e $q > 1$. A classe residual módulo q do elemento $a \in \mathbb{Z}$ pode ser assim definida:*

$$\bar{a} = \{x \in \mathbb{Z} \text{ tal que } x \equiv a(mod\ q)\}$$

Exemplo 2.2

Se $m = 2$, temos 2 classes residuais : $\bar{0} = \{\ldots -4, -2, 0, 2, 4, \ldots\}$ *e* $\bar{1} = \{\ldots -3, -1, 1, 3, \ldots\}$
Se $m = 3$, temos 3 classes residuais : $\bar{0}, \bar{1}, \bar{2}$
Se $m = 5$, temos 4 classes residuais : $\bar{0}, \bar{1}, \bar{2}, \bar{3}, \bar{4}$

Definição 2.3 Conjunto das Classes Residuais *Seja $\mathbb{Z}_q = \{\bar{0}, \bar{1}, \ldots, \overline{q-1}\}$ o conjunto das classes residuais dos inteiros módulo q.*

Notemos que $\mathbb{Z}_q$ tem exatamente q elementos, portanto $\mathbb{Z}_q$ é finito.

Propriedade 1 *Se $a \equiv b(mod\ q)$, então $\bar{a} = \bar{b}$.*

Com o objetivo de determinar quais são os $\mathbb{Z}_q$ que são corpos, vamos apresentar mais alguns conceitos em $\mathbb{Z}$.

Definição 2.4 Máximo Divisor Comum *Seja a e $b \in \mathbb{Z}$ com $a \neq 0$ ou $b \neq 0$. O MDC de a e b é $d \in \mathbb{Z}$, se as condições forem verdadeiras:*

(i) d é um divisor comum de a e b, ou seja, $d \mid a$ e $d \mid b$

(ii) d é divisível por todo divisor comum de a e b, ou seja, $\exists c \in \mathbb{Z},\ c \mid a$ e $c \mid b \implies c \mid d$.

Lema 2.1 Divisão Euclidiana *Dados $a \in \mathbb{Z}$, $b \in \mathbb{Z}$ e $b \neq 0$, $\exists c$, $\exists r$ inteiros únicos $\in \mathbb{Z}$ tais que $a = bc + r$ e $0 \leq r < |b|$. Então o $MDC(a, b) = MDC(b, r)$.*

c e r são chamados, respectivamente, o quociente e resto da divisão de a por b. O algoritmo euclidiano, um método eficiente para calcular o MDC [27], usa o lemma 2.1.

Teorema 2.1 Lema de Euclides *Sejam a e $b \in \mathbb{Z}$ não-nulos e seja $d = MDC(a, b)$. Então, $\exists \lambda, \mu \in \mathbb{Z}$ tais que $d = \lambda a + \mu b$.*

Proposição 1 *Dois inteiros a e b são primos entre si se, e somente se, $\exists \lambda, \exists \mu \in \mathbb{Z}$ tais que $\lambda a + \mu b = 1$.*

Prova Se a e b são primos entre si, então $MDC(a,b) = 1$ e pelo lema 2.1, temos que $\exists\lambda,\ \exists\mu \in \mathbb{Z}$ tais que $MDC(a,b) = \lambda a + \mu b$. Então:

$MDC(a,b) = \lambda a + \mu b = 1\ (multiplica - se\ ambos\ os\ lados\ por\ d = MDC(a,b))$
$d(\lambda a + \mu b) = d$

Se $\lambda a + \mu b = 0$, então $d = 0$, o que é um absurdo.
Se $\lambda a + \mu b < 0$, então $d < 0$ e $\lambda a + \mu b > 0$, o que é um absurdo.
Se $\lambda a + \mu b > 0$, vamos chamar esse resultado de $d^{'}$.

$$dd^{'} = d$$

Nesse caso, sabemos que $d^{'} = 1$.
Suponhamos que $\exists\lambda,\ \exists\mu \in \mathbb{Z}$ tais que $\lambda a + \mu b = 1$.

$$\lambda a + \mu b = 1\ (divide - se\ ambos\ os\ lados\ por\ d = MDC(a,b))$$
$$\frac{\lambda a + \mu b}{d} = \frac{1}{d}$$
$$\frac{\lambda a}{d} + \frac{\mu b}{d} = \frac{1}{d}$$

Sabemos que $MDC(a,b) \mid a$ e $MDC(a,b) \mid b$. Portanto, $\frac{a}{d}, \frac{b}{d} \in \mathbb{Z}$. Podemos chamar $\frac{a}{d}$ de c e $\frac{b}{d}$ de d.

$$\lambda c + \mu d = \frac{1}{d}$$

Pelo lema 2.1, então $\frac{1}{d}$ é o $MDC(c,d) \in \mathbb{Z}$. Nesse caso, sabemos que $d = 1$. □

Proposição 2 *Para $[a] \in \mathbb{Z}_q$ é invertível se, e somente se, $MDC(a,q) = 1$.*

Prova Suponhamos que $[a]$ seja inversível. Assim, $\exists b \in \mathbb{Z}$ tal que $[a].[b] = 1$. Assim, $[a.b] = 1$ e $a.b \equiv 1\ mod\ q$. Isso implica que $\exists s$ tal que $ab + sq = 1$. Pela proposição 1, implica que $MDC(a,q) = 1$.

Suponhamos que $MDC(a,m) = 1$. Pela proposição 1, $\exists b, c \in \mathbb{Z}$ tais que $ba + cm = 1$. Assim, $b.a \equiv 1\ mod\ q$. Portanto, $[a].[b] = [a.b] = 1$. □

2.2 Espaço Vetorial, Grupo, Anel e Corpo de Galois

Definição 2.5 Espaço Vetorial *Um espaço vetorial $\mathbb{V}$ é um conjunto não vazio de objetos chamados vetores e as suas duas operações chamadas adição e multiplicação com números reais, que seguem 10 propriedades. As propriedades são válidas para $\forall u, v, w \in V$ e $a, b \in \mathbb{R}$.*

(i) a *soma* $v + u \in V$

(ii) $u + v = v + u$

(iii) $(u + v) + w = u + (v + w)$

(iv) $\exists 0 \in V$ *um vetor zero tal que* $u + 0 = u$

(v) $\forall u \in V \exists -u \in V$ *um vetor tal que* $u + (-u) = 0$

(vi) $a.u \in V$

(vii) $a.(u + v) = a.u + a.v$

(viii) $(a + b).u = a.u + b.u$

(ix) $a(b.u) = (ab).u$

(x) $1.u = u$

Definição 2.6 Subespaço Vetorial *Dado um espaço vetorial* $\mathbb{V}$*, um subconjunto* W *não vazio é um subespaço vetorial de* V *se:*

(i) $\forall u, v \in W$*, temos que* $u + v \in W$

(ii) $\forall a \in \mathbb{R}, \forall u \in W$*, temos que* $a.u \in W$

Definição 2.7 Independência Linear *Sejam* V *um espaço vetorial e* $v_1, \ldots, v_n \in V$*. Dizemos que o conjunto* $\{v_1, \ldots, v_n\}$ *é linearmente independente, se a equação* $a_1v_1 + \ldots + a_nv_n = 0$ *implica que* $a_1 = a_2 = \ldots = a_n = 0$*. Se* $\exists a_i \neq 0$*, dizemos que* $\{v_1, \ldots, v_n\}$ *é linearmente dependente.*

Definição 2.8 Combinação Linear *Sejam* V *um espaço vetorial,* $v_1, \ldots, v_n \in V$ *e* $a_1, \ldots a_n \in \mathbb{Z}$*. O vetor da forma* $v = a_1v_1 + \ldots + a_nv_n$ *é um elemento de* V *e o chamamos de combinação linear de* $v_1, \ldots, v_n$*.*

Teorema 2.2 Dependência Linear *O conjunto* $\{v_1, \ldots, v_n\}$ *é linearmente dependente se, e somente se,* $\exists v_i$ *tal que* v_i *é uma combinação linear de* $v_j, j \neq i, 1 \leq j \leq n$*.*

Definição 2.9 Operação Binária $\mathbb{G}$ *é um conjunto não vazio,* $\phi : \mathbb{G}\ X\ \mathbb{G} \to \mathbb{G}$*, é uma função.* ϕ *é chamada operação binária tal que* $\phi(a, b) = a + b$ *ou* $\phi(a, b) = a.b$*. Considere as seguintes propriedades:*

Associatividade *Se* $a, b, c \in G$ *então* $a + (b + c) = (a + b) + c$*. Se* $a, b, c \in G$ *então* $a.(b.c) = (a.b).c$*.*

Elemento Neutro $\exists 0 \in G$ *tal que se* $a \in G$ $0 + a = a + 0 = a$. $\exists e \in G$ *tal que se* $a \in G$ $e.a = a.e = a$.

Elemento Inverso *Se* $a \in G, \exists b \in G$ *tal que* $a+b = b+a = 0$. b *é escrito como* $b = -a$. *Se* $a \in G, \exists b \in G$ *tal que* $a.b = b.a = e$. b *é escrito como* $b = a^{-1}$.

Comutatividade *Se* $a, b \in G$ *então* $a + b = b + a$. *Se* $a, b \in G$ *então* $a.b = b.a$.

Definição 2.10 Grupo *Se* (G, ϕ) *é um grupo, então satisfaz as propriedades associatividade, elemento neutro e elemento inverso. Se* $\phi(a, b) = a + b$, *então* G *é um grupo aditivo. Se* $\phi(a, b) = a.b$, *então* G *é um grupo multiplicativo. Se o grupo* (G, ϕ) *satisfaz a propriedade de comutatividade, ele é um grupo abeliano ou comutativo.*

Definição 2.11 Classe de Conjugação *Dois elementos* $a, b \in G$ *são chamados conjugados, se* $\exists g \in G$ *tal que* $gag^{-1} = b$. *A classe de equivalência que contém o elemento* $a \in G$ *é* $Cl(a) = \{gag^{-1} : g \in G\}$ *e é chamado a classe de conjugação de* a.

Definição 2.12 *A é um grupo comutativo aditivo, não vazio, e* $\exists \varphi$ *uma segunda operação binária* $A\ X\ A \to A$ *chamada multiplicação. Considere as seguintes propriedades:*

Associatividade *Se* $a, b, c \in A$ *então* $a.(b.c) = (a.b).c$.

Distributividade da multiplicação com relação a adição *Se* $a, b, c \in A$ $a.(b+c) = (a.b) + (a.c)$ *e* $(b + c).a = (b.a) + (c.a)$.

Elemento Neutro $\exists 1 \in A$ *tal que se* $a \in G$ $1.a = a.1 = a$.

Comutatividade *Se* $a, b \in A$ *então* $a.b = b.a$.

Exemplo 2.3 *Para um dado número primo* p, *o conjunto de números inteiros* $\{0, 1, 2, ..., p-1\}$ *é um grupo comutativo com relação à adição módulo-p. O conjunto de números inteiros* $\{1, 2, ..., p-1\}$ *é um grupo comutativo com relação à multiplicação módulo-p.*

Definição 2.13 Anel *Se* (A, ϕ, φ) *é um anel, então satisfaz as propriedades associatividade, distributividade da multiplicação com relação a adição e elemento neutro. Se ele também satisfaz a propriedade comutatividade,* (A, ϕ, φ) *é um anel comutativo.*

Exemplo 2.4 *Como exemplos de anéis comutativos, temos os conjuntos dos números racionais* $\mathbb{Q}$, *reais* $\mathbb{R}$, *complexos* $\mathbb{C}$ *e e inteiros mod q (q é primo) sob as operações de adição e multiplicação usuais.*

Definição 2.14 Domínio de Integridade *Um anel* A *é chamado domínio de integridade, se possuir a propriedade:* $\forall a, b, \in A$ *e* $a \neq 0$ *e* $b \neq 0 \Longrightarrow a.b \neq 0$

$\oplus$	0	1
0	0	1
1	1	0

Tabela 2.1: Operação Adição $\oplus$ do Corpo de Galois $\mathbb{GF}_2$

$\otimes$	0	1
0	0	0
1	0	1

Tabela 2.2: Operação Multiplicação $\otimes$ do Corpo de Galois $\mathbb{GF}_2$

Exemplo 2.5 *Os anéis $\mathbb{Z}$, $\mathbb{Q}$, reais $\mathbb{R}$, $\mathbb{C}$ são todos domínios de integridade. O anel $\mathbb{Z}_q$ é um domínio de integridade, quando q é um número primo.*

Teorema 2.3 *O anel $\mathbb{Z}_q$ é um corpo se, e somente se, q é um número primo.*

Prova Se $\mathbb{Z}_q$ é um corpo se, e somente se, todos os seus elementos $\bar{0}, \bar{1}, \ldots, \overline{q-1}$ são inversíveis. Pela proposição 2, significa que $MDC(0,q) = MDC(1,q) = MDC(2,q) = \ldots = MDC(q-1,q) = 1$. Portanto, q é primo. □

Definição 2.15 Corpo *Um anel onde todo elemento não nulo é invertível é chamado de Corpo.*

Exemplo 2.6 *O Corpo de Galois $\mathbb{GF}_2$ é o conjunto $\mathbb{A} = \{0,1\}$ e as operações $+$ e $.$. O $\mathbb{GF}_8$ ilustrado nas tabelas 2.3 e 2.4.*

Teorema 2.4 $\mathbb{GF}_q$ *Seja $\mathbb{GF}_q$ um corpo finito com q elementos. Então:*

(i) $q = p^n$ para algum primo p e para algum inteiro positivo n,

$\oplus$	0	1	2	3	4	5	6	7
0	0	1	2	3	4	5	8	7
1	1	2	3	4	5	6	7	0
2	2	3	4	5	6	7	0	1
3	3	4	5	6	7	0	1	2
4	4	5	6	7	0	1	2	3
5	5	6	7	0	1	2	3	4
6	6	7	0	1	2	3	4	5
7	7	0	1	2	3	4	5	6

Tabela 2.3: Operação Adição $\oplus$ do $\mathbb{GF}_8$

$\otimes$	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6	7
2	0	2	4	6	0	2	4	6
3	0	3	6	1	4	7	2	5
4	0	4	0	4	0	4	0	4
5	0	5	2	7	4	1	6	3
6	0	6	4	2	0	6	4	2
7	0	7	6	5	4	3	2	1

Tabela 2.4: Operação Multiplicação $\otimes$ do $\mathbb{GF}_8$

(ii) $\mathbb{GF}_q$ *contém o sub-corpo* $\mathbb{GF}_p$,

(iii) $\mathbb{GF}_q$ *é um espaço vetorial sob* $\mathbb{GF}_p$ *de dimensão* n,

(iv) $p\alpha = 0$ *para todo* $\alpha \in \mathbb{GF}_q$ *e*

(v) Isomorfismo: Dados dois corpos finitos C_q *e* G_q*, ambos com* q *elementos, existe uma única bijeção* $f : C_q \to G_q$ *com* $f(a+b) = f(a) + f(b)$ *e* $f(a.b) = f(a).f(b)$ *para* $\forall a, b \in C_q$*. Dizemos que* C_q *é isomorfo a* G_q*.*

Teorema 2.5 *Seja* $\mathbb{GF}_q$ *um corpo finito e* $a \in \mathbb{GF}_q$ *não nulo. Então* $a^{q-1} = 1$.

Prova Seja $b_1, b_2, \ldots, b_{q-1}$ os $q-1$ elementos não nulos de $\mathbb{GF}_q$. Os $q-1$ números $a.b_1$, $a.b_2 \ldots a.b_{q-1}$ são todos não nulos e distintos. Sabemos que pela operação multiplicação

$$(a.b_1).(a.b_2).\ldots.(a.b_{q-1}) = b_1.b_2.\ldots.b_{q-1}$$
$$a^{q-1}.(b_1.b_2.\ldots.b_{q-1}) = b_1.b_2.\ldots.b_{q-1}$$

Como $a \neq 0$ e $(b_1.b_2 \ldots b_{q-1}) \neq 0$, logo $a^{q-1} = 1$. $\square$

Definição 2.16 Ordem do Corpo *O número de elementos de um corpo finito* G *é denominado ordem de* G*. Um corpo de Galois de ordem* q *é representado por* $\mathbb{GF}_q$*. Um corpo finito* G *tem ordem* p^n*, onde* p *é a característica do corpo* G *e* $n = [K : Z_p]$.

Proposição 3 *Todo o corpo finito tem* p^n *elementos para algum primo* p *e algum* $n \in \mathbb{Z}^+$.

Prova

Proposição 4 *Para cada primo* p *e para cada* $n \in \mathbb{Z}^+$*, existe um corpo com* p^n *elementos.*

Prova

Proposição 5 *Qualquer corpo com p^i elementos é isomorfo à extensão de decomposição de $x^q - x, q = p^i$ sobre $\mathbb{GF}_q$.*

Prova

Exemplo 2.7

$$\begin{aligned}
&\textit{Corpo de Galois de ordem } 3: \ \mathbb{GF}_3 = \{0, 1, 2\} \\
&\textit{Corpo de Galois de ordem } 4: \ \mathbb{GF}_3 = \{0, 1, 2, 3\} \\
&\textit{Corpo de Galois de ordem } 5: \ \mathbb{GF}_5 = \{0, 1, 2, 3, 4\} \\
&\textit{Corpo de Galois de ordem } 7: \ \mathbb{GF}_7 = \{0, 1, 2, 3, 4, 5, 6\}
\end{aligned}$$

Definição 2.17 Ordem do Elemento *Seja $a \in \mathbb{GF}_q$ não nulo. A ordem de a, representada por $ord(a)$, é o menor inteiro positivo o tal que $a^o = 1$.*

Propriedade 2 *Se elevarmos ambos os lados a potência $q-1$, temos que $(a^o)^{q-1} = 1$. Multiplicando ambos os lados por a^o, temos que $(a^o)^q = a^o$ e substituindo a^o por x, temos $x^q = x$. Portanto, todos os $a \in \mathbb{GF}_q$ satisfazem a equação: $x^q - x = 0, q = p^n$.*

Teorema 2.6 *Seja $\mathbb{GF}_q$ um corpo finito e $a \in \mathbb{GF}_q$ não nulo. Seja o a ordem de a. Então $o \mid q-1$.*

Prova Suponhamos que $o \nmid q-1$. Assim, $q - 1 = ko + r$, onde $0 < r < o$.

$$a^{q-1} = a^{ko+r} = a^{ko}.a^r = (a^o)^k.a^r$$

Pelo teorema 2.5, $a^{q-1} = 1$ e $a^o = 1$, então $a^r = 1$. Isso é um absurdo, pois $0 < r < o$ e pela definição 2.17, o é o menor inteiro positivo tal que $a^o = 1$. Portanto, $o \mid q-1$. □

Definição 2.18 Adição módulo-m *Seja $a, b \in \mathbb{GF}_q$ não nulos. A operação adição módulo-m é definida como:*

$$\begin{aligned}
a \oplus b &= c \\
c &= (a + b)(mod\ m)
\end{aligned}$$

Isso é, a adição de quaisquer dois elementos $a, b \in \mathbb{GF}_q$ é o resto da divisão da adição aritmética $(a + b)$ por m.

Definição 2.19 Multiplicação módulo-m *Seja $a, b \in \mathbb{GF}_q$ não nulos. A operação multiplicação módulo-m é definida como:*

$$\begin{aligned}
a \otimes b &= c \\
c &= (a.b)(mod\ m)
\end{aligned}$$

Isso é, a multiplicação de quaisquer dois elementos $a, b \in \mathbb{GF}_q$ é o resto da divisão da multiplicação aritmética $(a.b)$ por m.

Definição 2.20 *Dado um corpo $\mathbb{GF}_q$, as operações adição e multiplicação seguem as seguintes propriedades:*

(i) $\mathbb{GF}_q$ *é um anel comutativo com relação a operação de adição. O elemento* 0 *é o elemento neutro.*

(ii) $\mathbb{GF}_q^*$ *é um anel comutativo com relação a operação de multiplicação. O elemento* 1 *é o elemento neutro.*

(iii) a operação multiplicação é distributiva com relação a adição:

$$a \otimes (b \oplus c) \;=\; (a \otimes b) \oplus (a \otimes c)$$

Definição 2.21 Elemento Primitivo *Seja $a \in \mathbb{GF}_q$ não nulo. a é chamado elemento primitivo se a ordem de a é $q-1$. Todos os corpos tem um elemento primitivo.*

Proposição 6 *Todas potências do elemento primitivo geram todos os elementos não nulos de $\mathbb{GF}_q$.*

Prova

Proposição 7 *Todos os corpos tem pelo menos um elemento primitivo.*

Prova

Proposição 8 *Cada elemento $b \in \mathbb{GF}_q$ não nulo pode ser representado através de um conjunto com potências de um elemento primitivo, com o expoente variando de* 1 *até* $q-1$.

Prova Pela definição 2.17, $b^o \;=\; 1$, onde o é a ordem de b. Pelo teorema 2.6, $o \mid q-1$. Pelo teorema 2.6, $b^{q-1} \;=\; 1$. Seja α o elemento primitivo de $\mathbb{GF}_q$.

$$\begin{gathered} a^i, i = 1, a = a \\ a^{q-1} \;=\; 1 \\ b^{q-1}.a^{q-1} \;=\; b^{q-1} \end{gathered}$$

Teorema 2.7 Pequeno Teorema de Fermat *Se p é um número primo e $p \nmid a$, então $a^{p-1} \equiv 1 (mod\ p)$ para $\forall a \in \mathbb{Z}$. Além disso, $a^p \equiv a (mod\ p)$. Portanto, $p \mid a^p \;-\; a$. [27]*

Prova Seja $P(n)$ a proposição que $p \mid n^p - n$ para $\forall n \in \mathbb{Z}$ e para p primo.

Base: em $P(1)$ é verdadeira, pois $1^p - 1$ é divisível por p.

Passo de indução: Suponhamos que P(k) é verdadeira $\forall k \in \mathbb{Z}$ e para p primo.

Queremos provar, para $k+1$, que P(k+1) é verdadeira. Podemos escrever $(k+1)^p$ da seguinte forma:

$$(k+1)^p = k^p + ip + 1, \text{ para } i \in \mathbb{Z}$$

Subtraindo $(k+1)$ em ambos os lados, temos que:

$$(k+1)^p - (k+1) = ip + (k^p - k)$$

Como $(k^p - k)$ é divisível por p por hipótese, temos que $(k+1)^p - (k+1)$ é divisível por p. □

2.3 Anéis de Polinômios

Definição 2.22 Anel de Polinômios *Dado um corpo $\mathbb{GF}_p$ definimos o anel comutativo com unidade $\mathbb{GF}_p[x]$ como sendo o conjunto das expressões da forma $P[x] = a_o + a_1x + a_2x^2 + \ldots + a_{n-1}x^{n-1} + a_nx^n$, com $a_n \neq 0$ e $n \in \mathbb{Z}^+$ chamados de polinômios com coeficientes em $\mathbb{GF}_p$. Cada polinômio define uma função polinomial $p(c) : \mathbb{GF}_p \to \mathbb{GF}_p$, $c \longmapsto p(c) = a_o + a_1c + a_2c^2 + \ldots + a_nc^n$. A distinção entre um polinômio e uma função polinomial é bem ilustrada pelo polinômio $x^p - x$, $x \in \mathbb{Z}$. Pelo teorema 2.7, a função polinomial $p(x) = x^p - x$ tem propriedades diferentes do polinômio equivalente, onde para $\forall x \in \mathbb{Z}$, temos que $P[x] = 0$.*

Teorema 2.8 *Sejam f e g dois polinômios não nulos em $\mathbb{GF}_p[x]$. Então:*

(i) fg é um polinômio não nulo

(ii) o grau de $f.g$ é igual a soma do grau de f mais o grau de g

(iii) fg é um polinômio de grau 1 se, e somente se, f e g são polinômios de grau 1

(iv) fg é um polinômio de grau 0 se, e somente se, f e g são polinômios de grau 0

(v) se $f + g \neq 0$, $grau(f+g) \leq max(grau(f), grau(g))$

Prova Suponhamos que f tenha grau m e que g tenha grau n. Se $k \in \mathbb{Z}^+$, temos que

$$(fg)_{m+n} = \sum_{i=0}^{m+n+k} f_i g_{m+n+k-i}$$

Sabemos que $i \leq m$ e $m + n + k - i \leq n$. Assim é necessário que $m + k \leq i \leq m$, que implica que $k = 0$ e $i = m$. Então

$$(fg)_{m+n} \ = \ f_m g_n$$

e

$$(fg)_{m+n+k} \ = \ 0$$

Corolário 2.9 *Suponhamos que f, g e h são polinômios em* $\mathbb{GF}_p[x]$ *tais que* $f \neq 0$ *e* $fg \ = \ fh$. *Então* $g \ = \ h$.

Prova Suponhamos que $fg \ = \ fh$.

$$f(g \ - \ h) \ = \ 0$$

Como $f \neq 0$, por (i) temos que $g \ - \ h \ = \ 0$. □

Lema 2.2 *Suponhamos que f e d sejam polinômios não nulos em* $\mathbb{GF}_p[x]$ *tais que* $grau(d) \leq grau(f)$. *Então* $\exists g$ *polinômio em* $\mathbb{GF}_p[x]$ *tal que ou* $f - dg \ = \ 0$ *ou* $grau(f - dg) < grau(f)$

Prova Suponhamos que

$$f \ = \ a_m x^m \ + \ \sum_{i=0}^{m-1} a_i x^i, \ a_m \neq 0$$

$$d \ = \ b_n x^n \ + \ \sum_{i=0}^{n-1} b_i x^i, \ b_n \neq 0$$

Seja m o grau de f e n o grau de d. Então $m > n$. Suponhamos que $g \ = \ \frac{f}{d}$. Vamos tomar o polinômio $g^{'}$ igual ao termo de g de maior grau:

$$g \ = \ \tfrac{a_m}{b_n} x^{m-n} \ + \ \ldots$$
$$g^{'} \ = \ \tfrac{a_m}{b_n} x^{m-n}$$

Se tomarmos $g \ = \ g^{'} \ = \ \frac{a_m}{b_n} x^{m-n}$,

$$f - \tfrac{a_m}{b_n} x^{m-n} \ = \ 0 \ ou$$
$$grau(f - \tfrac{a_m}{b_n} x^{m-n}) < grau(f)$$

□

Usando o lemma 2.2, podemos mostrar que a operação de divisão de polinômios pode ser feita em $\mathbb{GF}_p[x]$.

Definição 2.23 *Seja $f, g \in \mathbb{GF}_p[x]$, f e g polinômios e g não nulo. Existem um par de polinômios únicos q e r tal que $f(x) = q(x)g(x) + r(x)$, com $grau(r(x)) < grau(g(x))$.*

Definição 2.24 Polinômio Irredutível *Seja $p(x)$ um polinômio de grau m sobre $\mathbb{GF}_2$. Se $p(x)$ não for divisível por nenhum polinômio de grau $\leq m-1$, então $p(x)$ é irredutível sobre $\mathbb{GF}_2$.*

Proposição 9 *$\forall$ polinômio irredutível sobre $\mathbb{GF}_2$ de grau m divide $x^{2^m-1}+1$.*

Definição 2.25 Polinômio Primitivo *Seja $p(x)$ um polinômio de grau m sobre $\mathbb{GF}_2$. Se 2^m-1 for o menor inteiro positivo para o qual $p(x)$ divide $x^{2^m-1}+1$, então $p(x)$ é um polinômio primitivo de grau m sobre $\mathbb{GF}_2$.*

Definição 2.26 Polinômio Redutível e Irredutível *Dado um corpo $\mathbb{GF}_p[x]$, um polinômio f é redutível em $\mathbb{GF}_p[x]$, se existem os polinômios g, h em $\mathbb{GF}_p[x]$ de grau ≥ 1 tais que $f = gh$ e f e senão, f é irredutível em $\mathbb{GF}_p[x]$.*

Proposição 10 *Seja $F[x]$ um polinômio irredutível de grau n em $GP_p[x]$. Então $GP_p[x]/F[x]$ é um corpo finito com p^n elementos.*

Teorema 2.10 *Seja $P[x]$ um irredutível polinômio sob $\mathbb{GP}_p$ e cujo grau é m. Então o conjunto de todos os polinômios em x de grau $m-1$ e coefientes $\in \mathbb{GP}_p$ com as operações adição e multiplicação módulo-2 é um corpo de ordem p_m.*

2.4 Polinômios sobre $\mathbb{GF}_q$

Códigos de bloco são construídos com elementos de um corpo $\mathbb{GF}_q$ onde q ou é primo p ou uma potência de p. Os dados de sistemas de transmissão e armazenamento de dados podem ser facilmente codificados em códigos com símbolos gerados a partir de um corpo $\mathbb{GF}_2$ ou $\mathbb{GF}_{2^m}$ [31].

Definição 2.27 Código Binário *Código binário é um código em que o número de palavras de código é 2^m, ou seja, $q = 2$.*

O corpo $\mathbb{GF}_{2^m}$ consiste de todos os polinômios P em α de grau $\leq m-1$ com coeficientes em $\mathbb{GF}_2$. Um elemento em $\mathbb{GF}_{2^m}$ é da forma:

$$a_0 a_1 \ldots a_{m-1} \leftrightarrow a_0 + a_1\alpha + \ldots + a_{m-1}\alpha^{m-1}$$

onde $a_i \in \mathbb{GF}_2$ e $P(\alpha) = 0$

Na representação polinomial, o elemento primitivo α é a raiz do polinômio primitivo $p(x)$ de grau m. Essa representação é usada para a operação adição de dois elementos de $\mathbb{GF}_{2^m}$.

Exemplo 2.8 Representação Polinomial do Corpo $\mathbb{GF}_{2^4}$ *Considere o corpo $\mathbb{GF}_{2^4}$ definido pelo polinômio primitivo $p(x) = 1+x^3+x^4$. O corpo tem $2^4 = 16$ elementos. Esses serão os símbolos usados pelo código. A representação binária é expressa pela pela tupla (x^4, x^3, x^1, x^0). Na representação exponencial, o elemento nulo e os 4 primeiros elementos não nulos $\alpha, \alpha^2, \alpha^3, \alpha^4$ são expressos de forma trivial. Para obter a representação dos outros elementos $\alpha^5, \ldots, \alpha^{14}$ é necessário usar o polinômio primitivo. Para isso, define-se α raiz do polinômio primitivo:* [1]

$$\begin{aligned} p(\alpha) &= 0 \\ 1 + \alpha + \alpha^4 &= 0 \\ \alpha^4 &= -1 - \alpha = 1 + \alpha \end{aligned}$$

Para se obter os elementos:

$$\begin{aligned} \alpha^5 &= \alpha.\alpha^4 = \alpha.(1+\alpha) = \alpha + \alpha^2 \\ \alpha^6 &= \alpha.\alpha^5 = \alpha^2 + \alpha^3 \\ \alpha^7 &= \alpha.\alpha^6 = \alpha^3 + \alpha^4 = 1 + \alpha + \alpha^3 \\ \alpha^8 &= \alpha.\alpha^7 = \alpha + \alpha^2 + \alpha^4 = \alpha + \alpha^2 + 1 + \alpha = 1 + \alpha^2 \\ \alpha^9 &= \alpha.\alpha^8 = \alpha + \alpha^3 \\ \alpha^{10} &= \alpha.\alpha^9 = \alpha^2 + \alpha^4 \;=\; \alpha^2 + 1 + \alpha \;=\; 1 + \alpha + \alpha^2 \\ \alpha^{11} &= \alpha.\alpha^{10} = \alpha + \alpha^2 + \alpha^3 \\ \alpha^{12} &= \alpha.\alpha^{11} = \alpha^2 + \alpha^3 + \alpha^4 = 1 + \alpha + \alpha^2 + \alpha^3 \\ \alpha^{13} &= \alpha.\alpha^{12} = \alpha + \alpha^2 + \alpha^3 + \alpha^4 = \alpha + \alpha^2 + \alpha^3 + 1 + \alpha = 1 + \alpha^2 + \alpha^3 \\ \alpha^{14} &= \alpha.\alpha^{13} = \alpha + \alpha^3 + \alpha^4 = \alpha + \alpha^3 + 1 + \alpha = 1 + \alpha^3 \\ \alpha^{15} &= \alpha.\alpha^{14} = \alpha^0 \end{aligned}$$

O corpo $\mathbb{GF}_{256}$ é um espaço vetorial sob $\mathbb{GF}_2$ de dimensão 8 e é muito usado para construir códigos de bloco. A aritmética binária utiliza adição e multiplicação módulo-2. Um polinômio $F[x]$ definido sobre $\mathbb{GF}_2$ possui a forma:

$$\begin{aligned} P[x] \;=\; a_o \;+\; a_1 x \;+\; a_2 x^2 \;+ \ldots + \; a_{n-1} x^{n-1} \;+\; a_n x^n \\ \text{com } a_i \in \{0,1\} \text{ e } n \in \{0,1,2,3,4,5,6,7\} \end{aligned}$$

2.5 Construção de $\mathbb{GF}_q$ (Galois Estendido)

Definição 2.28 Representação Exponencial *Seja α um elemento primitivo em $\mathbb{GF}_q, q = 2^m$. Então α tem ordem $q-1$ e as potências distintas de α $\{1, \alpha, \alpha^2, \ldots \alpha^{q-2}\}$ geram todos*

[1] *Na aritmética binária, -1= +1.*

representação binária	*representação polinomial*	*representação exponencial*
0000	0	0
1000	1	1
0100	α	α
0010	α^2	α^2
0001	α^3	α^3
1100	$\alpha + 1$	α^4
0110	$\alpha + \alpha^2$	α^5
0011	$\alpha^2 + \alpha^3$	α^6
1101	$1 + \alpha + \alpha^3$	α^7
1010	$1 + \alpha^2$	α^8
0101	$\alpha + \alpha^3$	α^9
1110	$1 + \alpha + \alpha^2$	α^{10}
0111	$\alpha + \alpha^2 + \alpha^3$	α^{11}
1111	$1 + \alpha + \alpha^2 + \alpha^3$	α^{12}
1011	$1 + \alpha^2 + \alpha^3$	α^{13}
1001	$1 + \alpha^3$	α^{14}

Tabela 2.5: Mapeamento dos elementos do corpo $\mathbb{GF}_4$ gerado pelo polinômio $1 + x^3 + x^4$

os elementos diferentes de 0 *de* $\mathbb{GF}_q$.

A representação exponencial é utilizada na operação multiplicação de dois elementos do $\mathbb{GF}_q$, *pois* $\alpha^i.\alpha^j = \alpha^{i+j}$.

Definição 2.29 *Um polinômio* $p(x)$ *irredutível sobre* $\mathbb{GF}_q$ *de grau* m *é um polinômio primitivo se suas raízes forem um elemento de* $\mathbb{GF}_q$.

Definição 2.30 *Seja* $p(x)$ *um polinômio com coeficientes em* $\mathbb{GF}_2$. *Seja* $\beta \in \mathbb{GF}_q$. *Se* β *é uma raiz de* $p(x)$, *então* $\forall l \geq 0$, β^{2^l} *é também uma raiz de* $p(x)$. *O elemento* β^{2^l} *é chamado conjugado de* β.

Teorema 2.11 Raízes do Polinômio *Os* $2^m - 1$ *elementos diferentes de* $0 \in \mathbb{GF}_q$ *são as raízes do polinômio* $x^{2^m-1} + 1$.

Corolário 2.12 *Para todos os elementos* $\in \mathbb{GF}_q$, *incluindo o elemento* 0, *esses são as raízes do polinômio* $x^{2^m-1} + x$.

Definição 2.31 Polinômio Minimal *Seja* $\beta \in \mathbb{GF}_q$. *O polinômio minimal* $\phi(x)$ *de* β *é o polinômio de menor grau com coeficientes em* $\mathbb{GF}_2$ *tal que* $\phi(\beta) = 0$.

Teorema 2.13 *Seja $\phi(x)$ o polinômio minimal de um elemento $\beta \in \mathbb{GF}_q$ e e o menor inteiro tal que $\beta^{2^e} = \beta$. Então $\phi(x) = \prod_{i=0}^{e-1}(x + \beta^{2^i})$*

Definição 2.32 Polinômio Gerador *Seja $C : (m, k)$ um código cíclico sobre $\mathbb{GF}_q$. $\exists g(x)$ de grau 1, tal que uma m-tupla $c \in C : (m, k)$ é uma palavra-código se, e somente se, $g(x) \mid c$.*

Propriedade 3 *O polinômio gerador é único. O grau do polinômio gerador é $m - k$ e seu grau é igual ao número de bits de paridade. $g(x)$ é o polinômio código de menor grau entre todos os polinômios código. O polinômio gerador é um divisor de $x^m - 1$. Todo polinómio código é um múltiplo do polinômio gerador.*

Proposição 11 *O polinómio código não nulo de grau mínimo de um determinado código cíclico $C : (m, k)$ é único.*

Prova Esse polinômio terá a forma $g(X) = g_0 + g_1X + \ldots + X^r$. Se existir outro polinômio com grau mínimo, este polinômio será da forma $g1(X) = g_{10} + g_{11}X + \ldots + X^r$. No entanto, porque o código $C : (m, k)$ é um código de bloco linear, a soma destes dois códigos polinómios deverá pertencer ao código, e esta soma nos fornece um polinômio de grau $(r-1)$, o que contradiz a hipótese que o grau mínimo possível é r. logo, o polinômio código não nulo de grau mínimo de um determinado código cíclico $C : (m, k)$ é único. □

Definição 2.33 *O polinômio gerador $g(x)$ é da forma $g[x] = 1 + g_1x + g_2x^2 + \ldots + g_{m-k}x^{m-k}$.*

Proposição 12 *Seja $a(x) = ao + a_1x + a_2x^2 + \ldots + a_{k-1}x^{k-1}$. Então, $(a(x)g(x) \; mod \; (x^m - 1)) \in C(m, k)$.*

Prova $a(x)g(x)$ pode ser escrito da seguinte forma:

$$\underbrace{a_o g(x)}_{\in C} + \underbrace{a_1 g(x)}_{\in C} + \ldots + \underbrace{a_{k-1}x^{k-1}g(x)}_{\in C}$$

Portanto, $a(x)g(x) \in C$.

Definição 2.34 Código de Blocos Linear *Um código de bloco de tamanho m e 2^k palavras mensagem é denominado um código de blocos linear $C : (m, k)$, se as 2^k palavras código formam um subespaço vetorial S de dimensão k, contido no espaço de vetores formado por todos os vetores de tamanho m, V_m, com componentes no $\mathbb{GF}_2$. Um código é linear, quando o elemento nulo pertence ao código e a soma de duas palavras código também é uma palavra código.*

Definição 2.35 Matriz Geradora Não Sistemática *Seja $C : (m, k)$ um código de blocos linear. Existem k vetores linearmente independentes tal que cada palavra código c pode ser escrita como uma combinação linear deles.*

$$c = v_0 g_0 \oplus v_1 g_1 \oplus \ldots \oplus v_{k-1} g_{k-1}$$

Esses vetores podem ser organizados em um matriz geradora G

$$G = \begin{bmatrix} g_0 \\ g_1 \\ \vdots \\ g_{k-1} \end{bmatrix} = \begin{bmatrix} g_{0,0} & g_{0,1} & \cdots & g_{0,m-1} \\ g_{1,0} & g_{1,1} & \cdots & g_{1,m-1} \\ \vdots & \vdots & \vdots & \vdots \\ g_{k-1,0} & g_{k-1,1} & \cdots & g_{k-1,m-1} \end{bmatrix}$$

Todo vetor v, a mensagem a ser enviada ou armazenada pode ser obtido através da multiplicação de matrizes:

$$\mathbf{c} = \mathbf{vG} = (v_0, v_1, \ldots, v_{k-1}) \begin{bmatrix} g_{0,0} & g_{0,1} & \cdots & g_{0,m-1} \\ g_{1,0} & g_{1,1} & \cdots & g_{1,m-1} \\ \vdots & \vdots & \vdots & \vdots \\ g_{k-1,0} & g_{k-1,1} & \cdots & g_{k-1,m-1} \end{bmatrix}$$

$$= (v_0, v_1, \ldots, v_{k-1}) \begin{bmatrix} g_0 \\ g_1 \\ \vdots \\ g_{k-1} \end{bmatrix} = v_0 g_0 \oplus v_1 g_1 \oplus \ldots \oplus v_{k-1} g_{k-1}$$

As linhas da matriz geradora G geram o código de blocos linear $C : (m, k)$, ou seja, as k linhas linearmente independentes de G definem completamente o código.

Definição 2.36 Matriz Geradora Sistemática *Um código de blocos linear e sistemático $C : (m, k)$ é especificado de forma única por uma matriz geradora da forma:*

$$G = \begin{bmatrix} g_0 \\ g_1 \\ \vdots \\ g_{k-1} \end{bmatrix} = \begin{bmatrix} p_{0,0} & p_{0,1} & \cdots & p_{0,m-k-1} & 1 & 0 & 0 & \ldots & 0 \\ p_{1,0} & p_{1,1} & \cdots & p_{1,m-k-1} & 0 & 1 & 0 & \ldots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ p_{k-1,0} & p_{k-1,1} & \cdots & p_{k-1,m-k-1} & 0 & 0 & 0 & \ldots & 1 \end{bmatrix}$$

ou, de uma forma mais compacta:

$$\mathbf{G} = [\mathbf{P}\mathbf{I}_\mathbf{k}]$$

onde P é a matriz de paridade.

Definição 2.37 Matriz de Verificação de Paridade *Em um código de blocos linear e sistemático $C : (m, k)$, as 2^k palavras código formam um subespaço vetorial S associado a um subespaço dual S_d do mesmo espaço V_m que é gerado pela matriz de verificação de paridade H. Cada vetor linha da matriz G é ortogonal às linhas da matriz H e vice-versa. Assim, cada código tem um dual que é o código gerado pela matriz H. Este código é um código $(n, n-k)$. A forma sistemática da matriz de paridade H do código C, gerado pela matriz G é:*

$$\mathbf{H} = \begin{bmatrix} 1 & 0 & 0 & \dots & 0 & p_{0,0} & p_{1,0} & \cdots & p_{k-1,0} \\ 0 & 1 & 0 & \dots & 0 & p_{0,1} & p_{1,1} & \cdots & p_{k-1,1} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & 1 & p_{0,m-k-1} & p_{1,m-k-1} & \cdots & p_{k-1,m-k-1} \end{bmatrix} = \left[\mathbf{I_{n-k}} \mathbf{P^T}\right]$$

onde P^T é a transposta da matriz de paridade P. A matriz H é construída tal que

$$\mathbf{G H^T = 0}$$

Definição 2.38 Síndrome de Detecção de Erro *Após a recepção ou a leitura da palavra código c, temos o vetor $r = r_0 r_1 \dots r_{m-1} \in \mathbb{GF}_2$. O vetor r pode ser diferente do vetor c transmitido ou armazenado. Seja o vetor $e = e_0 e_1 \dots e_{m-1} \in \mathbb{GF}_2$ tal que $e = r \oplus c$. Um mecanismo de detecção de erro pode ser implementado tal que $cH^T = 0$, o vetor síndrome:*

$$S = rH^T = (s_o s_1 \dots s_{m-k-1}) = (c \oplus e)H^T = cH^T \oplus eH^T = eH^T$$

Se o padrão de erro for igual a uma palavra código, o erro não será detectado. Existem $2^k - 1$ padrões de erros não detectáveis. O vetor síndrome pode ser calculado da forma:

$$\begin{aligned} s_0 &= e0 \oplus e_{m-k} \otimes p_{0,0} \oplus e_{m-k+1} \otimes e_{1,0} \oplus \dots \oplus e_{m-1} \otimes p_{k-1,0} \\ s_1 &= e1 \oplus e_{m-k} \otimes p_{0,1} \oplus e_{m-k+1} \otimes e_{1,1} \oplus \dots \oplus e_{m-1} \otimes p_{k-1,1} \\ &\vdots \\ s_{n-k-1} &= e_{m-k-1} \oplus e_{m-k} \otimes p_{0,m-k+1} \oplus e_{m-k+1} \otimes e_{1,m-k+1} \oplus \dots \oplus e_{m-1} \otimes p_{k-1,m-k+1} \end{aligned}$$

Exemplo 2.9 Código Hamming *No seu artigo [64], R. Hamming propôs uma codificação que levou o seu nome. Para $\forall m \geq 3$, $\exists\, CH : (n, k)$, um código hamming com as seguintes características: comprimento da palavra código é $n = 2^m - 1$, número de bits de dados na mensagem é $k = 2^m - m - 1$, número de bits de paridade é $m = n - k$, capacidade de correção de erros é $t = 1$, distância mínima é $d_{min} = 3$ e capacidade de*

e_0	e_1	e_2	e_3	e_4	e_5	e_6
0	*0*	*1*	*0*	*0*	*0*	*0*
1	*1*	*1*	*1*	*0*	*0*	*0*
1	*0*	*0*	*0*	*0*	*0*	*1*
0	*1*	*0*	*1*	*0*	*0*	*1*
0	*0*	*0*	*1*	*0*	*1*	*0*
1	*1*	*0*	*0*	*0*	*1*	*0*
0	*1*	*1*	*0*	*0*	*1*	*1*
1	*0*	*1*	*1*	*0*	*1*	*1*
0	*1*	*0*	*0*	*1*	*0*	*0*
1	*0*	*0*	*1*	*1*	*0*	*0*
1	*1*	*1*	*0*	*1*	*0*	*1*
0	*0*	*1*	*1*	*1*	*0*	*1*
1	*0*	*1*	*0*	*1*	*1*	*0*
0	*1*	*1*	*1*	*1*	*1*	*0*
1	*1*	*0*	*1*	*1*	*1*	*1*
0	*0*	*0*	*0*	*1*	*1*	*1*

Tabela 2.6: Padrões de erros para Exemplo 2.9

detecção de erros é $l = 2$. *Para* $m = 3$, $n = 7$, $k = 4$ *e para a matriz geradora* $\mathbf{G}$, *a matriz* $\mathbf{H}$ *desse* $CH : (7, 4)$ *é:*

$$\mathbf{G} = \mathbf{PI_4} = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 \end{bmatrix} \mathbf{H} = [\mathbf{I_{n-k}} \mathbf{P_T}] \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{bmatrix}$$

O vetor síndrome $S = (s_0, s_1, s_2) = (e_0, e_1, e_2, e_3, e_4 e_5 e_6)\mathbf{H^T} =$. *Então*

$$s_0 = e_0 \oplus e_3 \oplus e_5 \oplus e_6$$
$$s_0 = e_1 \oplus e_3 \oplus e_4 \oplus e_5$$
$$s_0 = e_2 \oplus e_4 \oplus e_5 \oplus e_6$$

Uma mensagem inicial (1010) *foi enviada ou armazenada como* $c = (1010)\mathbf{G} = (0011010)$ *e foi recebida ou lida como* $r = (0001010)$. *O vetor síndrome é* $S = (001)$. *Existem* 2^4 *padrões de erro que satisfazem esse sistema de equações que podem ser vistos na tabela 2.6.*

Para canais como o canal simétrico binário (BSC), o padrão de erros com menor número de erros é considerado o padrão correto e ele segue uma distribuição binomial.

$$0 = e_0 \oplus e_3 \oplus e_5 \oplus e_6$$
$$0 = e_1 \oplus e_3 \oplus e_4 \oplus e_5$$
$$1 = e_2 \oplus e_4 \oplus e_5 \oplus e_6$$

Assim, podemos calcular $c = r \oplus e = (0001010) \oplus (0010000) = (0011010)$. *Como utilizamos a matriz geradora sistemática, a mensagem inicial é* (1010).

2.6 Construção de um Código Corretor de Erros

O objetivo da codificação de canal é aumentar a resistência do sistema de comunicações digital face aos efeitos do ruído de canal. No caso particular dos códigos de bloco, a cada bloco de k bits da sequência binária gerada pela fonte faz-se corresponder um bloco de m bits (palavra de código) com $m > k$. Este processo de codificação deve ser concebido de modo que a decodificação tenha solução única. Note-se que do universo de $2m$ blocos binários de comprimento m apenas $2k$ são palavras de código (as que correspondem numa relação de um para um aos blocos binários de comprimento k gerados pela fonte). Este esquema está representado simbolicamente na Figura 2.1. 8.4. No caso de a transmissão se efetivar sem erros, o processo de decodificação conduz ao bloco de comprimento k que havia sido gerado pela fonte. Quando ocorrem erros de transmissão, a palavra de comprimento m recebida pode não ser uma palavra de código e o erro é detectado e/ou corrigido. Figura 8.4: Codificação de canal

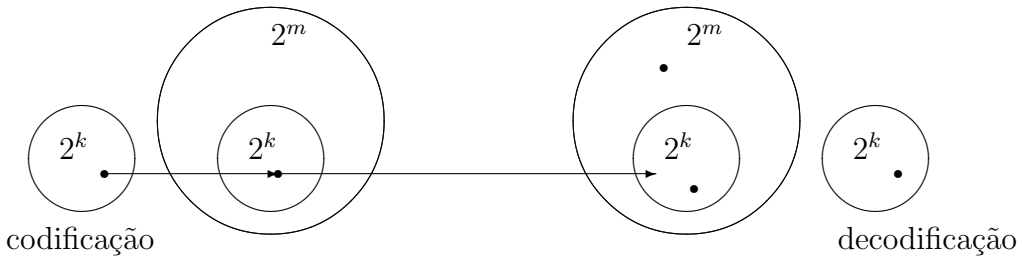

Figura 2.1: Codificação de canal

É de especial interesse o caso de $\mathbb{GF}_q, q = 2^m,\ m = 8$, quando cada símbolo representa

1 byte. Um byte representa 8 bits, que é a menor palavra de dados usualmente encontrada em sistemas de computadores.

O corpo $\mathbb{GF}_{2^8-1}$ usado para construir a codificação RS implementada no Hadoop consiste de polinômios $P[x]$ com coeficientes binários e grau até 7? e as operações de adição e multiplicação, calculadas módulo o polinômio irredutível $P[x] = x^8 + x^4 + x^3 + x^2 + 1$.

Capítulo 3

Codificação por Apagamento

One can no longer claim to be a communication or computer system engineer without knowing something about coding.
D. J. Costello, J. Hagenauer, H, Imai, S. B. Wicker [3]

A transmissão e o armazenamento de dados tem muito em comum. Ambos transferem dados da fonte para o destino. Para garantir confiabilidade nessas operações, é utilizada codificação por apagamento ou códigos corretores de erros. A Teoria dos Códigos tem sido estudada há décadas: por matemáticos nas décadas de 50 e 60 e a partir da década de 70 por engenheiros [12, 58].

Com a popularização dos computadores e as pesquisas espaciais, os códigos corretores tornaram-se parte comum de comunicações por satélite, de redes de computadores, de armazenamento em discos óticos e outros meios magnéticos. A presença dos códigos corretores de erros é frequente em nosso cotidiano: quando se assiste a um programa de televisão, quando se ouve música a partir de um CD, quando se faz um telefonema, quando se assiste um filme gravado em DVD, quando se navega pela internet.

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 [16] sem sobrecarregar a taxa de transmissão de informação ou o *overhead* de armazenamento [31].

Um dos principais parâmetros de um código para detecção de erros é a probabilidade de detecção de erro.

A probabilidade de erro no canal determina a capacidade de transferência de informação no canal. Os modelos estudados por pesquisadores envolvem canais simétricos, assimétricos e outros, com ou sem memória [41].

3.1 Shannon: conceitos e teoremas fundamentais

Shannon introduziu dois conceitos fundamentais sobre informação que é transmitida em um sistema de comunicação [67]:

a incerteza da informação se o dado que nos interessa é determinístico, então ele não tem valor algum. Por exemplo, a transmissão contínua de uma imagem em um sistema de televisão é supérflua. Desta forma, a fonte de informação é modelada por uma variável ou processo aleatório e uma probabilidade é utilizada para se desenvolver a teoria da informação.

a informação transmitida é digital o dado que nos interessa deve ser convertido em *bits* e ser entregue no destino corretamente, sem referência ao seu significado inicial. O trabalho do Shannon [16] parece ser o primeiro trabalho publicado que usa o termo *bit*.

Nesse mesmo trabalho, Shannon demonstrou dois importantes teoremas que são fundamentais na comunicação ponto-a-ponto:

source coding theorem introduz a entropia como medida da informação que, nesse caso, é caracterizada por a taxa mínima de código que representa uma informação livre de erros. Este teorema é a base téorica para compressão de dados.

channel coding theorem fala da capacidade de um canal com ruídos, na qual a informação é transmitida de forma confiável, desde que ritmo de transferência de dados seja menor que a capacidade do canal.

A probabilidade de erro no canal determina a capacidade de transferência de informação no canal.

3.2 Canais: modelos e erros

O código da fonte é o conjunto de elementos que definem forma como a informação será transmitida, por exemplo, em termos de *bits* e o código do canal inclui o código da fonte e a redundância (por exemplo, em termos de *bits*) introduzida para garantir a correção de erros.

Uma dificuldade encontrada por quem estuda códigos corretores de erros é que não existe uma nomenclatura unificada [69]. Também segundo [32], existem poucos pesquisadores que são programadores de sistemas e que fazem propostas nesse tema.

A idéia básica da código ótimo é que o objeto original possa ser reconstruído a partir de quaisquer k únicos fragmentos que são aproximadamente do mesmo tamanho do objeto original [14].

Corrigir erros é uma tarefa mais complexa que detectá-los. Detectar erros tem a mesma complexidade que a operação de codificar, que pode ser linear no tamanho das palavras código. A operação de decodificar para uma correção de erros ótima é um problema NP-difícil e são conhecidos apenas algoritmos eficientes para algumas classes de códigos. Novas classes de códigos com eficientes decodificadores e novos algoritmos para decodificação para códigos conhecidos são promissoras pesquisas [60].

Em canais binários simétricos (BSC), assume-se que ambos os erros $0 \rightarrow 1$ e $1 \rightarrow 0$ ocorrem com igual probabilidade [41].

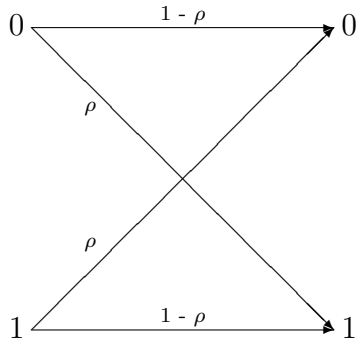

Figura 3.1: Canal binário simétrico [41]

ρ = probabilidade da ocorrência de um erro

Em algumas aplicações como comunicações óticas, os erros tem uma natureza assimétrica. Canais, onde esse tipo de erro ocorrem, podem ser modelados para canais binários assimétricos ou canal-Z, onde apenas erros com 1's ocorrem. Um exemplo disso

são sistemas que utilizam *photons* para transmitir informação [41].

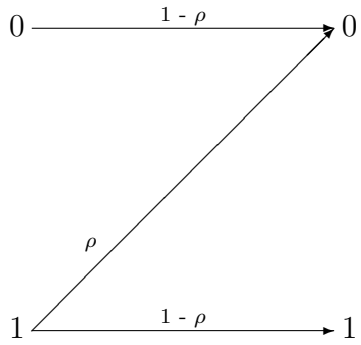

Figura 3.2: Canal binário assimétrico [41]

ρ = probabilidade da ocorrência de um erro

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 *Automatic Repeat reQuest*, 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 e não é necessário um caminho de retorno.

3.2.1 *Automatic Repeat reQuest* - ARQ

ARQ utiliza redundância para detectar erros em mensagens e, após a detecção, o destinatário solicita uma repetição da transmissão. Um caminho de retorno é necessário. São sistemas de *two-way transmission*. Exemplos de sistemas que utilizam ARQ são linhas telefônicas e alguns sistemas de satélite [31].

Se existe um grande atraso de propagação, uma grande distância entre emissor e destinatário, este método pode ser muito ineficiente. Podem existir casos em que a retransmissão não é possível, quando não existe *backup*.

Técnicas ARQ incluem repetição seletiva, *Go-back N* e sistemas de reconhecimento positivo ou negativo [65].

3.2.2 *Forward Correction Code* - FEC

Técnicas FEC representam um *one-way system*. A transmissão ou gravação é restrita a uma direção: da fonte para o sumidouro (destino). Sistemas de armazenamento de fita magnética e sistemas da *NASA's Deep Space Network* utilizam técnicas FEC [31].

O destinatário corrige a mensagem recebida através da codificação por apagamento. Este procedimento geralmente é chamado de correção de erros de repasse e pode ser implementado em *hardware* de propósito especial.

FEC utiliza redundância, assim o decodificador pode corrigir os erros de mensagens no destinatário. Uma analogia pode ser feita com uma pessoa que fala devagar e repetidamente, em uma linha telefônica com ruídos, acrescentando mais redundância para o ouvinte entender a mensagem correta.

Como FEC funcionam

Técnicas FEC incluem códigos de blocos (*block codes*), que são códigos sem memória e códigos convolacionais, que são códigos com memória [44].

3.3 Códigos de Blocos Lineares

Na Figura 3.3, vemos um sistema que utiliza código de blocos. A fonte envia uma sequência de dados para o codificador. O codificador divide esta sequência em 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 m *bits* chamada *codeword* ou palavra código representada por uma m-tupla de símbolos discretos $v = v_1, v_2, \ldots, v_m$. Os $m - k$ bits são os bits redundantes que provêm à codificação a capacidade de tratar os ruídos do canal.

Códigos de blocos são identificados pela notação (m, k), de acordo com o número de *bits* de saída m e o número de *bits* k de cada um dos blocos de entrada.

Todas as palavras código de um código de blocos tem tamanho fixo, que é um certo número de blocos de k *bits*.

A geração de uma palavra código depende apenas de um cálculo algébrico entre os k bits, portanto, um codificador pode ser implementado como um circuito lógico combinacional. O codificador executa o mapeamento: $T : U \rightarrow V$ onde U é um conjunto de

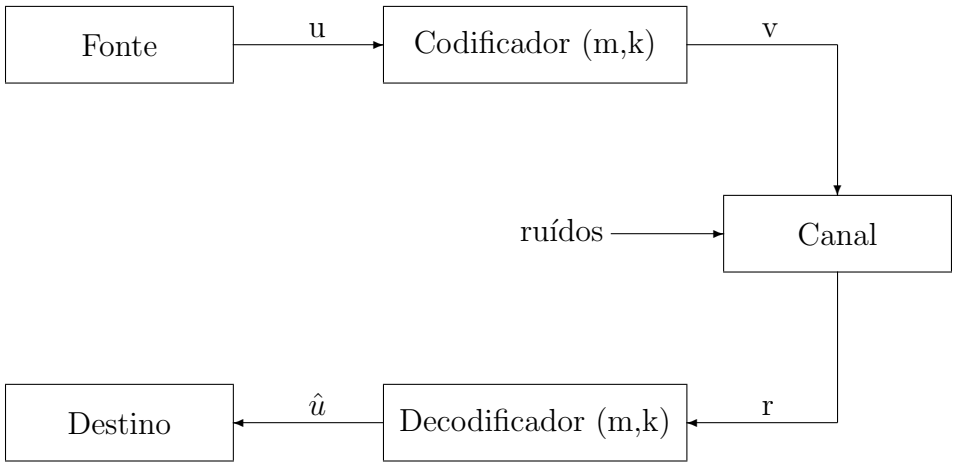

Figura 3.3: Códigos de bloco

palavras de dados de tamanho k e V é um conjunto de palavras código de tamanho m onde $m > k$. Cada uma das 2^k palavras de dados é mapeada para uma única palavra código.

A taxa de codificação e a sobrecarga de armazenamento são calculados a partir de k blocos originais [5, 17]. São gerados m símbolos pelo algoritmo de codificação. $R = \frac{k}{m}$ 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 a sobrecarga de armazenamento.

Se $k \leq m$, mais bits redundantes podem ser adicionados, com aumento do tamanho da palavra código, mantendo $R = \frac{k}{m}$ constante. Como escolher este número $m - k$ de bits redundantes para obter transmissão confiável em cima de um canal com ruídos é o problema principal do projeto do codificador. No destino, o decodificador extrai a sequência original de dados.

Outra métrica utilizada é a redundância que pode ser definida por $\frac{(m-k)}{m}$. A alta redundância reduz 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).

Existem vários códigos de blocos [1, 4] que são utilizados em redes e em sistemas de computadores. Alguns deles são códigos Reed-Solomon (RS), códigos Low-Density Parity-Check (LDPC) e códigos Tornado [40, 5].

Segundo [30], as principais características de códigos de blocos são, resumidamente:

taxa de codificação $R = \frac{k}{m}$ é a medida da eficiência do código, pois é o quociente do número de *bits* da palavra de dados sobre o número de *bits* total da palavra transmitida

distância mínima (d_{min}) é a menor distância de Hamming (definição 3.5) entre duas quaisquer palavras do código; ela depende do número de *bits* redundantes $q = m-k$, tal que ($d_{min} \leq q+1$); um código com $d_{min} = 1$ não tem capacidade de detectar erros

capacidade de detecção detecta até l erros, onde $l \leq d_{min} - 1$

capacidade de correção corrige os erros até t erros, onde $t \leq \lfloor \frac{d_{min}-1}{2} \rfloor$

capacidade de detecção e correção detecta até l erros e corrige os erros até t erros, onde $d_{min} \geq l + t + 1$ e $l > t$

Definição 3.1 *Seja uma sequência de bits $w = w_0 w_1 \ldots w_{k-1} \in \{0,1\}^k$. Podemos definir w como sendo um elemento $w = (w_0 w_2 \ldots w_{k-1})$ do produto cartesiano:*

$$\mathbb{Z}_2{}^k = \underbrace{\mathbb{Z}_2 \times \ldots \times \mathbb{Z}_2}_{k}$$

Um código C sobre o alfabeto $\mathbb{Z}$ possui três parâmetros fundamentais: $[n, M, d]$, onde n é seu comprimento no espaço vetorial $\mathbb{Z}^n$, M é o número de elementos de C e d é a distância mínima (definição 3.5) entre os elementos de C [58].

O conceito de equivalência de códigos também é importante na construção de um código.

Definição 3.2 Isometria *Seja A um alfabeto e $n \in \mathbb{Z}^*$ não nulo. Dizemos que uma função $F : A^n \to A^n$ é uma isometria de A^n, se ela preserva as distâncias de Hamming (definição 3.5) dos elementos de A^n, ou seja, $d(F(u), F(v)) = d(u, v),\ \forall u, v \in A^n$.*

Proposição 13 *Toda isometria de A^n é uma bijeção de A^n.*

Prova

Proposição 14 *(i) A função identidade de A^n é uma isometria.*

(ii) Se F é uma isometria de A^n, então F^{-1} é uma isometria de A^n.

(iii) Se F e G são isometrias de A^n, então $F \circ G$ é uma isometria de A^n.

Prova

Definição 3.3 *Dados dois corpos C e C' em $\mathbb{A}^n$, dizemos que C' é equivalente a C se $\exists F$ isometria de A^n tal que $F(C) = C'$.*

A equivalência de códigos é uma relação de equivalência , portanto possui as seguintes propriedades:

Propriedade 4 *(i) Reflexividade: todo código é equivalente a si próprio*

(ii) Simetria: se C' é equivalente a C, então C é equivalente a C'.

(iii) Transitividade: se C'' é equivalente a C' e C' é equivalente a C, então C'' é equivalente a C.

Algumas importantes isometrias para construção de códigos são as que apresentamos como exemplo:

Exemplo 3.1 *Se $f : A \to A$ é uma bijeção e dado um número $i \in \mathbb{Z}$ não nulo tal que $1 \leq i \leq n$, então*

$$\begin{aligned} {T_f}^i : A^n &\to A^n \\ (a_1, \ldots, a_i, \ldots, a_n) &\mapsto (a_1, \ldots, f(a_i), \ldots, a_n) \end{aligned}$$

é uma isometria.

Exemplo 3.2 *Se π é uma bijeção do conjunto $\{1, \ldots, n\}$ nele próprio (permutação), a aplicação de π dada por*

$$\begin{aligned} T_\pi : A^n &\to A^n \\ (a_1, \ldots, a_n) &\mapsto (a_{\pi(1)}, \ldots, a_{\pi(n)}) \end{aligned}$$

é uma isometria.

Apresentamos também um teorema sobre isometria relativa a métrica de Hamming (definição 3.5), importante para a construção do código da fonte e do código do canal, ambos citados na seção 3.2.

Teorema 3.1 *Seja $F : A^n \to A^n$ uma isometria, então $\exists \pi$ uma permutação do conjunto $\{1, \ldots, n\}$ e $\exists f_i$ bijeções de A, $i = 1, \ldots n$ tais que $F = T_\pi \circ T^1_{f_1} \circ \ldots \circ T^n_{f_n}$.*

Corolário 3.2 *Sejam C e C' dois códigos em A^n. Temos que C e C' são equivalentes se, e somente se, $\exists \pi$ uma permutação do conjunto $\{1, \ldots, n\}$ e $\exists f_i$ bijeções de A, $i = 1, \ldots n$ tais que*

$$C' = \{(f_{\pi(1)}(x_{\pi(1)}), \ldots, f_{\pi(n)}(x_{\pi(n)})), (x_1, \ldots, x_n) \in C\}$$

Definição 3.4 Peso Hamming *Seja* $u = u_0u_1 \dots u_k \in {\mathbb{Z}_2}^k$. *O peso Hamming de* u *é dado por*

$$\omega(u) = |\{i = 0, 1, \dots k-1 \ : u_i = 1\}|$$

Em outras palavras, $\omega(u)$ *é o número de* 1*'s entre os bits de* u.

Definição 3.5 Distância de Hamming *Seja* $u = u_0u_1 \dots u_k \in {\mathbb{Z}_2}^k$ *e* $v = v_0v_1 \dots v_k \in {\mathbb{Z}_2}^k$. *A distância entre* u *e* v *é dada por* $\delta(u, v) = |\{i = 0, 1, \dots k-1 \ : u_j \neq v_j\}|$ *Em outras palavras, a distância de Hamming* $\delta(u, v)$ *é o número de pares de bits correspondentes de* u *e* v *em que um bit difere do outro.*

Proposição 15 *Suponha que* $u, v \in {\mathbb{Z}_2}^k$. *Então* $\delta(u, v) = \omega(u \oplus v)$.

Prova Seja $P(n)$ a proposição que $\delta(u, v) = \omega(u \oplus v)$ para $\forall u, v \in {\mathbb{Z}_2}^n$.

Base: $P(1)$ é verdadeira, pois $\delta(0, 0) = \omega(0 \oplus 0) = 0$, $\delta(0, 1) = \omega(0 \oplus 1) = 1$, $\delta(1, 0) = \omega(1 \oplus 0) = 1$ e $\delta(1, 1) = \omega(1 \oplus 1) = 0$.

Passo de indução: Suponhamos que P(k) é verdadeira $\forall (u, v) \in {\mathbb{Z}_2}^k$ e para $\forall k < n$.

Queremos provar, para $k + 1$, que P(k+1) é verdadeira. Seja $u = u_0u_1 \dots u_k \in {\mathbb{Z}_2}^k$, $v = v_0v_1 \dots v_k \in {\mathbb{Z}_2}^k$, $u' = u_0u_1 \dots u_ku_{k+1} \in {\mathbb{Z}_2}^{k+1}$ e $v' = v_0v_1 \dots v_kv_{k+1} \in {\mathbb{Z}_2}^{k+1}$. Podemos escrever $\delta(u', v')$ como:

$$\delta(u', v') = \delta(u, v) + \delta(u_{k+1}, v_{k+1}).$$

Sabemos calcular $\delta(u_{k+1}, v_{k+1})$, pois $P(1)$ é verdadeira. Pela hipótese de indução, $\delta(u, v) = \omega(u \oplus v)$. Então, $\delta(u', v') = \omega(u, v) \oplus \omega(u_{k+1}, v_{k+1}) = \omega(u', v')\square$

Proposição 16 *Suponha que* $u, v \in {\mathbb{Z}_2}^k$. *Então* $\omega(u \oplus v) \leq \omega(u) + \omega(v)$

Prova Seja $P(n)$ a proposição que $\omega(u, v) \leq \omega(u) + \omega(v)$ para $\forall u, v \in {\mathbb{Z}_2}^n$.

Base: $P(2)$ é verdadeira,

u, v	$u \oplus v$	$\omega(u+v)$	$\leq \omega(u) + \omega(v)$
$00, 00$	00	0	0
$00, 01$	01	1	$0+1=1$
$00, 10$	10	1	$0+1=1$
$00, 11$	11	0	0
$01, 00$	01	1	$0+0=0$
$01, 01$	00	0	$1+1=2$
$01, 10$	10	1	$1+1=2$
$01, 11$	10	1	$1+0=1$
$10, 00$	10	1	$1+0=1$
$10, 01$	11	1	$1+1=2$
$10, 10$	00	0	$1+1=2$
$10, 11$	01	1	$1+0=1$
$11, 00$	11	0	$0+0=0$
$11, 01$	01	1	$0+1=1$
$11, 10$	01	1	$0+1=1$
$11, 11$	00	0	$0+0=0$

Passo de indução: Suponhamos que P(k) é verdadeira $\forall (u, v) \in {\mathbb{Z}_2}^k$ e para $\forall k < n$. Queremos provar, para $k+1$, que P(k+1) é verdadeira. Seja $u = u_0 u_1 \dots u_k \in {\mathbb{Z}_2}^k$, $v = v_0 v_1 \dots v_k \in {\mathbb{Z}_2}^k$ e $z = u \oplus v = z_0 z_1 \dots z_k \in {\mathbb{Z}_2}^k$ e $u^{'} = u_0 u_1 \dots u_k u_{k+1} \in {\mathbb{Z}_2}^k$, $v^{'} = v_0 v_1 \dots v_k v_{k+1} \in {\mathbb{Z}_2}^k$ e $z^{'} = u \oplus v = z_0 z_1 \dots z_k z_{k+1} \in {\mathbb{Z}_2}^{k+1}$. Podemos escrever $\omega(u^{'} \oplus v^{'}) = \omega(z^{'})$ como

$$\omega(z^{'}) = \omega(z) + \omega(u_{k+1} \oplus v_{k+1}) = \omega(u \oplus v) + \omega(u_{k+1} \oplus v_{k+1})$$

Sabemos calcular $\omega(u_{k+1} \oplus v_{k+1})$, pois $P(2)$ é verdadeira. Então $\omega(u_{k+1} \oplus v_{k+1}) \leq \omega(u_{k+1}) + \omega(v_{k+1})$. Pela hipótese de indução, $\omega(u \oplus v) \leq \omega(u) + \omega(v)$. Então

$$\omega(z^{'}) = \omega(u \oplus v) + \omega(u_{k+1} \oplus v_{k+1}) \leq \omega(u) + \omega(v) + \omega(u_{k+1}) + \omega(v_{k+1})$$

□

Proposição 17 *A distância* $\delta : {\mathbb{Z}_2}^k \times {\mathbb{Z}_2}^k \to \mathbb{Z}^*$ *tem as seguintes propriedades para* $\forall u, v, z \in {\mathbb{Z}_2}^k$*:*

(i) $\delta(u, v) \geq 0$

(ii) $\delta(u, v) = 0$, *se, e somente se,* $u = v$

(iii) $\delta(u, v) = \delta(v, u)$

(iv) $\delta(u, z) \leq \delta(u, v) + \delta(v, z)$

Prova

Definição 3.6 Distância Mínima *Seja $C : (n, k)$ um código de blocos linear. A distância mínima desse código é dada por*

$$\begin{aligned} d_{min} &= min\{\delta(u, v), u, v \in C : (n, k), u \neq v\} \\ &= min = \{\omega(u), u \in C : (n, k), u \neq 0\} \end{aligned}$$

o valor mínimo do peso das palavras código não nulas desse código.

Definição 3.7 *Seja $k \in \mathbb{Z}^*$ e $u \in {\mathbb{Z}_2}^k$. O conjunto $B(u, k) = \{v \in {\mathbb{Z}_2}^k : \delta(u, v) \leq k\}$*

Teorema 3.3 *Seja $k, m \in \mathbb{Z}^*$ e $n > k$, a função de codificação $\alpha : {\mathbb{Z}_2}^k \to {\mathbb{Z}_2}^m$, $\zeta = \alpha({\mathbb{Z}_2}^k)$ e $\tau(c) =$ sequência de bits de c após a leitura ou a transmissão de c*

(i) Suponhamos que $\delta(c, d) > k$ para $\forall c, d \in \zeta, c \neq d$. Então a transmissão ou o armazenamento de $\delta(c, \tau(c)) \leq k$ pode ser sempre detectada, ou seja, até k erros podem ser detectados;

(ii) Suponhamos que $\delta(u, v) > 2k$ para $\forall c, d \in \zeta, c \neq d$. Então a transmissão ou o armazenamento de $\delta(c, \tau(c)) \leq k$ pode ser sempre detectada e corrigida, ou seja, até $2k$ erros podem ser detectados e corrigidos.

Prova (i) Já que $\delta(c, d) > k$ para $\forall c, d \in \zeta, c \neq d$, então $\forall c \in \zeta$, $B(c, k) \cap \zeta = c$. Assim, com a transmissão ou a leitura de no mínimo 1 e no máximo k erros de c, temos que $\tau(c) \neq c$ e $\tau(c) \in B(c, k)$. Logo, $\tau(c) \notin \zeta$.

(ii) Como (i) é verdadeira, $\forall c, d \in \zeta, d \neq c$, temos que, pela proposição 16, $2k < \delta(c, d) \leq \delta(c, \tau(c)) + \delta(\tau(c), d)$. Sabemos que $\delta(c, \tau(c)) \leq k$ e que $\delta(d, \tau(c)) > k$, de modo que $\tau(c) \notin B(c, k)$. Por isso sabemos exatamente qual elemento de ζ que originou $\tau(c)$.

Teorema 3.4 *Seja $C : (n, k)$ um código de blocos linear e sua matriz de verificação de paridade $\mathbf{H}$. Para cada vetor código $u \in C : (n, k)$ e l igual do seu peso $\omega(u)$, $\exists l$ colunas da matriz $\mathbf{H}$ tais que o vetor soma dessas l colunas é igual ao vetor 0. Da mesma forma, se o vetor soma de l colunas da matriz $\mathbf{H}$ for o vetor 0, então $\exists v \in C : (n, k)$ tal que $\omega(v) = l$.*

Prova

Corolário 3.5 *Seja $C : (n, k)$ um código de blocos linear e sua matriz de verificação de paridade* **H**. *O peso mínimo ou a distância mínima de $C : (n, k)$ é igual ao menor número de colunas de* **H** *cuja soma é o vetor* 0.

Códigos Reed-Solomon são códigos de blocos, lineares e cíclicos. São códigos parametrizáveis, cuja capacidade de correção de erros pode ser alterada facilmente.

Segundo [38], códigos 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 onde o conjunto de símbolos de entrada é consideravelmente grande.

O sistema de armazenamento OceanStore [4] e o protocolo BitTorrent (aplicação da camada de rede da internet) usam uma codificação RS.

Redundant Arrays of Inexpensive [Independent] Disks (RAID) é uma classe de códigos RS. RAID é um método para prover tolerância a falhas ou alto desempenho em sistemas de armazenagem utilizando para isso uma codificação de correção de erros ou paridade. RAID foi introduzido por D. A. Patterson na Universidade da California, Berkeley (UC Berkeley) em 1988 [26].

Segundo [42], 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.

São conceitos básicos [13]:

data striping é uma técnica para segmentar dados sequênciais, como um arquivo, de maneira que o acesso a segmentos sequênciais seja feito por diferentes dispositivos de armazenamento. Esta técnica é útil quando se quer processar mais rapidamente os pedidos de acesso a dados que os dispositivos de armazenamento permitem. Diferentes segmentos de dados são mantidos em diferentes dispositivos de armazenamento. A falha de um dos dispositivos torna toda a sequência de dados indisponível. Essa desvantagem é superada pelo armazenamento de informações redundantes (custo de armazenamento extra), como a paridade, com o objetivo de correção de erros. As configurações de RAID que utilizam paridade são RAID-2, RAID-3, RAID-4, RAID-5 e RAID-6 [46].

stripe são segmentos consecutivos ou faixas que são escritos sequencialmente através de cada um dos discos de um *array* ou conjunto. Cada segmento tem um tamanho definido em blocos.

RAID 2

Esta configuração divide os dados a nivel de *bit* e usa códigos Hamming para correção de erros. Por exemplo, o código Hamming(7,4) (quatro *bits* de dados e tres *bits* de paridade) permite usar 7 discos em RAID 2, sendo 4 usados para armazenar dados e 3 usados para correção de erros. Esta codificação tornou-se padrão para *hard drives* e tornou-se desnecessária, assim deixou de ser vantajosa.

RAID 3

Esta configuração divide os dados a nivel de *byte* com um disco apenas para paridade. Isto requer que todos os discos operem em *lockstep* (rotação de todos os discos em sincronismo). Assim como RAID-2, tornou-se obsoleta.

RAID 4

Esta configuração divide os dados a nivel de bloco com um disco apenas para paridade. Se o controlador de disco permitir, um conjunto RAID 4 pode atender várias solicitações de leitura ao mesmo tempo. Todos os *bits* de paridade estão em um único disco, o que pode se tornar um gargalo. RAID-5 substituiu esta configuração.

RAID 5

Esta configuração divide os dados a nivel de bloco com um único bloco de paridade por *stripe* e os blocos de paridade ficam distribuídos por todos os discos. Esta configuração privilegia a leitura. Uma síndrome é computada para permitir a perda de uma unidade. Essa síndrome P pode ser um simples XOR de dados pelos *stripes*.

RAID 6

Esta configuração divide os dados a nivel de bloco com dois blocos de paridade por *stripe* e os blocos de paridade distribuídos por todos os discos. Duas síndromes diferentes precisam ser computadas para permitir a perda de quaisquer duas unidades. Uma delas, P pode ser um simples XOR de dados pelos *stripes*, como em RAID 5. A outra, Q pode ser um XOR de um *linear feedback shift register* de cada *stripe* [24].

Códigos Low-Density Parity-Check e Tornado são códigos de blocos, lineares e acíclicos.

Código Low-Density Parity-Check (LPDC) é uma codificação baseada em grafos regulares [51]. Códigos LDPC são conhecidos também como códigos Gallager [47]. Uma aplicação dessa codificação é a rede *wireless* WMAN WiMAX (IEEE 802.16e *standard for microwave communications*) para internet móvel [48].

Códigos Tornado são uma classe de códigos LDPC (Low Density Parity Check) que utiliza grafos irregulares e que foi proposta por M. Luby [42]. Segundo [4], são mais rápidos para codificar e decodificar e necessitam de um pouco mais de k 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.

Em [11], os autores apresentam códigos Tornado baseados em grafos irregulares. Segundo [32], as implicações práticas desses códigos ainda não foram bem estudadas.

3.3.1 Capacidade de Correção de Erros

Após a transmissão ou armazenamento de um vetor código v, um padrão de l erros resultará em um vetor recebido ou lido r que difere do vetor v em l bits, ou seja, $\delta(v, r) = l$. Se a distância mínima de $C : (n, k)$ é d_{min}, nenhum padrão de erro de $d_{min} - 1$ ou menos pode trocar um vetor código por outro. Ou seja, todo padrão de erro de $d_{min} - 1$ ou menos irá resultar em um vetor r que não é um vetor código de C. Quando o receptor detecta que o vetor recebido não é um vetor código de C, os erros são detectados. Por esse motivo, um código de bloco com distância mínima d_{min} é capaz de detectar todos os padrões de erro com $d_{min} - 1$ ou menos. Esse mesmo código não é capaz de detectar todos os padrões de erro com d_{min} ou mais, pois existe pelo menos um par de vetores código que diferem de d_{min} bits.

Proposição 18 *Seja um código de bloco $C : (n, k)$. O código C é capaz de detectar $2^n - 2^k$ padrões de erro de tamanho n.*

Prova Existem $2^n - 1$ padrões de erro não nulos e desses $2^k - 1$ padrões de erro são idênticos aos $2^k - 1$ vetores código não nulos. Se um desses padrões ocorre, um vetor código v é recebido ou lido como w. A síndrome será igual a 0. O decodificador aceita w e confirma uma decodificação incorreta. Portanto existem $2^k - 1$ padrões de erro não detectáveis. Assim, existem $2^n - 1 - (2^k - 1) = 2^n - 2^k$ padrões de erro detectáveis, isso inclue o vetor 0.

Podemos acrescentar que um código de bloco $C : (n, k)$ corrige 2^{n-k} padrões de erro de tamanho n.

Definição 3.8 Distribuição Peso *Seja um código de bloco $C : (n, k)$. Seja A_i o número de vetores código de C com peso i. A sequência $A_o, A_1, \ldots A_n$ é a distribuição peso de C. No caso de canal binário simétrico, a probabilidade que o decodificador falhe em detectar erros pode ser expressada:*

$$P_u(E) = \sum_{i=1}^{n} A_i p^i (1-p)^{n-i}$$

onde p é probabilidade da ocorrência de um erro no canal. Se a distância mínima de C é d_{min}, $A_1 = A_{d_{min}-1} = 0$.

Exemplo 3.3 *Para o código $CH(7, 4)$ do exemplo 2.9, a distribuição peso é $A_0 = 1, A_1 = A_2 = 0, A_3 = 7, A_5 = A_6 = 0$ e $A_7 = 1$. A probabilidade de não detectar erros é:*

$$P_u(E) = 0 + 0 + 7p^3(1-p)^4 + 7p^4(1-p)^3 + 0 + 0 + p^7 = 7p^3(1-p)^4 + 7p^4(1-p)^3 + p^7$$

*Mesmo para $p \ll 1$, $p = 10^{-2}$, $P_u(E) \approx 7 \times 10^{-6}$. Esse resultado pode ser assim interpretado: a cada 1 milhão de palavras código, 7 erros não são detectados pelo decodificador do código $CH(7, 4)$. Devido a sua simplicidade, os códigos Hamming são usados em memórias RAM, onde a taxa de erros é baixa e são conhecidos pelo nome SECDED (*Single Error Correction, Double Error Detection*).*

Vamos determinar a capacidade de correção de erros de um código de bloco linear $C : (n, k)$.

Proposição 19 Capacidade de correção de erros de um código de bloco linear $C : (n, k)$ *Seja $C : (n, k)$ um código de bloco linear. Seja t o número de bits que pode ser corrigido e d_{min} a distância mínima de C. Como d_{min} é um inteiro, vamos escrever t na seguinte condição:*

$$2t + 1 \leq d_{min} \leq 2t + 2$$

Então, C é capaz de detectar todos os padrões de t erros.

Prova Seja v o vetor código transmitido ou armazenado e r o vetor recebido ou lido. Seja $w \in C$. As distâncias de Hamming dos vetores satisfazem a inequação:

$$\delta(v, r) + \delta(w, r) \geq \delta(v, w)$$

Suponhamos que ocorram $t^{'}$ erros na transmissão ou armazenagem de v, então $\delta(v, r) = t^{'}$. Como $v, w \in C$, $\delta(v, w) \geq d_{min} \geq 2t + 1$. Assim, reescrevendo a inequação:

$$\begin{aligned} \delta(v, r) + \delta(w, r) &\geq \delta(v, w) \\ t^{'} + \delta(w, r) &\geq 2t + 1 \\ \delta(w, r) &\geq 2t + 1 - t^{'} \end{aligned}$$

Se $t' \leq t$, então

$$\delta(w, r) > t$$

Isso mostra que na ocorrência de de t erros ou menos, o vetor r é muito próximo do vetor v que qualquer outra palavra w no código C.

O código C não é capaz de corrigir todos os padrões de erros com $l > t$. Um código de bloco linear com distância mínima d_{min} garante corrigir todos os padrões de erros $t = \lfloor \frac{(d_{min}-1)}{2} \rfloor$ ou menos. O parâmetro t é chamado capacidade de correção de erros aleatórios de um código.

Se um código de bloco linear $C : (n, k)$, capaz de corrigir todos os padrões de erro de peso t ou menor, é usado na transmissão ou armazenamento em um canal binário simétrico com probabilidade de erro p, a probabilidade do decodificador falhe em detectar erros tem limite superior dado por:

$$P_u(E) = \sum_{i=t+1}^{n} \binom{n}{i} p^i (1-p)^{n-i}$$

Esses códigos são usados de tal forma que os padrões de erro de peso λ são corrigidos e os padrões de erro de peso $l > \lambda$ são detectados. Isto é possível se $d_{min} \geq l + \lambda + 1$.

Teorema 3.6 Limite de Singleton *A distância mínima de um código de blocos $C : (n, k)$ tem limite superior dado por $d_{min} \leq n - k + 1$.*

Prova O vetor código não nulo com menor peso tem peso d_{min}, por definição. Existem vetores código com apenas um símbolo de informação não-nulo + $(n - k)$ símbolos de paridade. Tal vetor código não pode ter peso maior que $1 + (n - k)$. Logo, o peso mínimo do código - isto é, d_{min}, não pode ser maior do que $1 + (n - k)$.

Códigos de bloco que alcançam o limite de Singleton são chamados códigos MDS. Com códigos binários, a igualdade só se atinge com códigos de repetição. Nesse caso, a sobrecarga de armazenamento é grande ($k = 1$, $O = n/k = n$).

3.3.2 Matriz Padrão e Detecção da Síndrome

Um código de bloco linear $C : (n, k)$ é construído como uma bijeção entre os 2^k vetores mensagem (código da fonte) e o conjunto $c_1, c_2, \ldots c_{2k}$ (código do canal). Cada um desses vetores é transmitido através de um canal e convertido no vetor recebido r que pode ser qualquer vetor dos 2^n vetores do espaço vetorial V^n definido sobre o corpo binário GF_2. Toda decodificação é essencialmente uma regra de decisão, baseada no espaço vetorial V^n

particionado em 2^k conjuntos disjuntos $D_1, D_2, \ldots, D_{2^k}$ tal que o vetor $c_i \in D_i$. Existe uma correspondência única entre o conjunto D_i e o vetor c_i. Se o vetor recebido $r \in D_i$, ele será decodificado para c_i.

A construção da matriz Padrão é o método para executar a decodificação. A matriz é construída da seguinte forma:

1. A primeira linha é uma palavra código que começa com o vetor 0. Essa linha contém 2^k vetores do conjunto dos 2^n e tem a forma: $c_1 = (0, 0, \ldots, 0)c_2c_3 \ldots c_{2^k}$.

2. A segunda linha começa com um padrão de erro e_2, selecionado dos $2^n - 2k$ vetores restantes. Essa linha contém a soma dos 2^k vetores com e_2, ou seja, $c_i \oplus e_2,\ \forall i = 2 \ldots 2^k$.

3. O processo é repetido para cada uma das 2^{n-k} linhas da matriz padrão, selecionando 2^{n-k} padrões de erro dos vetores restantes.

Assim, a matriz apresenta a forma:

$$\begin{bmatrix} c_1 = (0, 0 \ldots 0) & c_2 & \ldots & c_i & \ldots & c_{2^k} \\ e_2 & c_2 \oplus e_2 & \ldots & c_i \oplus e_2 & \ldots & c_{2^k} \oplus e_2 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ e_{2^{n-k}} & c_2 \oplus e_{2^{n-k}} & \ldots & c_i \oplus e_{2^{n-k}} & \ldots & c_{2^k} \oplus e_{2^{n-k}} \end{bmatrix}$$

Nessa matriz, a soma de quaisquer dois vetores da mesma linha é um vetor código. Existem 2^{n-k} linhas distintas nessa matriz. Essas linhas são chamadas de *cosets* do código de bloco linear $C : (n, k)$. O vetor que começa cada um dos *cosets* é denominado o líder do *coset*, e pode ser qualquer vetor da linha.

Seja D_j a j-ésima coluna da matriz padrão da forma: $D_j = \{c_j,\ e_2 \oplus c_j,\ e_3 \oplus c_j, \ldots\ e_{2^{n-k}} \oplus c_j\}$ onde c_j é um vetor código de C e $e_2, e_3, \ldots e_{2^{n-k}}$ são *cosets* líderes. As 2^k colunas distintas $D_1, D_2, \ldots D_{2^k}$ podem ser usadas para decodificar um vetor r. Se o vetor c_j for transmitido pelo canal, pela definição de D_j, o vetor $c_j \in D_j$ se o padrão de erro obtido no canal é um *coset* líder. Dessa forma, o vetor r pode ser decodificado corretamente para o vetor c_j. Em outras palavras, se um padrão de erro do canal não é um *coset* líder, a decodificação resultará em erro.

Seja x um padrão de erro do canal da forma $x = e_l \oplus c_i$ com $2 \leq l \leq 2^{n-k}$ e $2 \leq i \leq 2^k$. Assim, o vetor recebido r pode ser escrito como $r = c_j \oplus x = e_j \oplus (c_i \oplus c_j) = e_l \oplus c_s$. O vetor $r \in D_s$ e é decodificado para c_s, que não é o vetor c_j. Isso resulta em erro. Portanto, a decodificação é correta se, e somente se, o padrão de erro obtido no canal é um *coset* líder.

Para minimizar a probabilidade de erro, todos os padrões de erro que se podem corrigir, que são os *cosets* líderes, terão de ser os padrões mais prováveis. Em caso de transmissão sobre um canal BSC, os padrões mais prováveis são os que possuem os menores pesos possíveis. Portanto, para construir a matriz padrão, os *cosets* líderes são escolhidos entre os vetores de menor peso que estão disponíveis. A decodificação nesse caso é a decodificação de distância mínima: o vetor código decodificado está à distância mínima com relação ao vetor recebido.

Teorema 3.7 *Todo código linear* $C : (n, k)$ *é capaz de corrigir* 2^{n-k} *padrões de erro.*

Teorema 3.8 *Para um código de bloco linear* $C : (n, k)$ *com distância mínima* d_{min}*, todos os vetores de peso* $t = \lfloor \frac{d_{min}-1}{2} \rfloor$ *ou menos, podem ser líderes do* coset *da matriz padrão de* C*. Se todos os vetores de peso* t *forem usados como líderes do* coset*, existe pelo menos um vetor de peso* $t + 1$ *que não pode ser usado como líder do* coset*.*

Teorema 3.9 *Todos os* 2^k *vetores de um* coset *tem a a mesma síndrome. As síndromes de diferentes* cosets *são distintas.*

Prova Tomando um líder do *coset* como o vetor e_i, todos os outros vetores desse *coset* são a soma do líder do *coset* e do vetor código c_i. Para esse caso, a síndrome é calculada como

$$S = (c_i \oplus e_i)H^T = c_i H^T \oplus e_i H^T = e_i H^T$$

Assim, podemos afirmar que a síndrome de qualquer vetor do *coset* é igual à síndrome do líder desse *coset*.

Síndromes são vetores com $(n - k)$ componentes que possuem bijeção com os *cosets*. Para cada padrão de erro corrigível, existe um vetor síndrome diferente. Isto permite-nos implementar uma decodificação simples através da construção de uma tabela onde os padrões de erro e os seus vetores síndrome correspondentes estão organizados de tal forma que, quando o decodificador realiza o cálculo de um vetor síndrome, pode reconhecer o padrão de erro correspondente. Assim, o decodificador é capaz de corrigir o vetor recebido, adicionando o padrão de erro ao vetor recebido.

A decodificação das síndromes consiste dos seguintes passos:

1. Computar $S = rH^T$
2. Uma tabela que mapea a bijeção entre S e e_l é usada para identificar o padrão de erro e_l. Então, e_l é o padrão de erro obtido no canal.

3. Decodificar o vetor r no vetor $v \in C$, tal que $v = r \oplus e_l$

Para $n - k$ grande, esse esquema de decodificação é impraticável.

Teorema 3.10 Limite de Hamming *O número de bits de verificação de paridade de qualquer código linear binário (n,k) com distância mínima dmin* $\geq 2t+1$ *satisfaz o limite de Hamming*

$$2^{n-k} \geq 1 + \binom{n}{1} + \binom{n}{2} + \ldots + \binom{n}{t} = \sum_{i=0}^{t} \binom{n}{t}$$

Prova Pela proposição 19, $d_{min} \geq 2t + 1$, então todos os vetores código de peso t ou menos (isto é, padrões de t erros ou menos) podem ser usados como *coset leaders* de uma matriz padrão. Se o código corrige t ou menos erros por vetor código, o conjunto de todos os padrões de t ou menos erros (incluindo o padrão nulo)

$$\binom{n}{0} + \binom{n}{1} + \binom{n}{2} + \ldots + \binom{n}{t}$$

deve ser menor ou igual ao número de *cosets leaders* 2^{n-k}.

O limite de Hamming estabelece um valor máximo para a capacidade de correção de erros t de um código. A igualdade verifica-se nos códigos perfeitos, onde não há nenhum vetor código de n bits a uma distância maior que t de outro vetor código.

Temos um limitante menor para a capacidade de correção de erros de um código, o limite de Plotkin.

Teorema 3.11 Limite de Plotkin *Se dispormos em as* $2k$ *vetores código de um código* $C(n, k)$*, obteremos uma matriz de* 2^k *linhas e n colunas. Em cada uma dessas colunas, metade dos bits (isto é,* $\frac{2^k}{2}$*) é* 0 *e a outra metade é* 1*. Logo, o número total de* 1*'s que existem em todos os vetores código é*

$$n\frac{2^k}{2} = n2^{k-1}$$

Como existem $2^k - 1$ *vetores não nulos em* $C(n, k)$*, o peso médio dos vetores código é*

$$\frac{n2^{k-1}}{2^k - 1}$$

Assim,

$$d_{min} \leq \frac{n2^{k-1}}{2^k - 1}$$

ou

$$\frac{d_{min}}{n} \leq \frac{2^{k-1}}{2^k - 1}$$

3.3.3 Exemplos de Códigos de Blocos

Definição 3.9 *Seja uma sequência de bits* $w = w_0 w_1 \dots w_{m-1} \in \{0,1\}^m$. *Podemos definir* w *como sendo um elemento* $w = (w_0 w_2 \dots w_{m-1})$ *do produto cartesiano:*

$$\mathbb{Z}_2{}^m = \underbrace{\mathbb{Z}_2 \times \dots \times \mathbb{Z}_2}_{m}$$

Exemplo 3.4 *Considere o código de bloco (6,5). A função de codificação* $\alpha : \mathbb{Z}_2{}^5 \to \mathbb{Z}_2{}^6$ *é definida da seguinte forma: para cada* $w \in \mathbb{Z}_2{}^5$, *seja* $c = \alpha(w) = w_0 w_1 \dots w_4 w_5 \in \mathbb{Z}_2{}^6$ *e* $w5 = w_0 + w_1 + \dots + w_4 \ (mod\ 2)$. *Cada* $c \in \mathbb{Z}_2{}^6$, *o número de bits* 1 *é sempre par. Desta forma, um erro (de transmissão ou de armazenamento) será detectado, mas sem a possibilidade de correção.*

Exemplo 3.5 *Considere o código de bloco (21,7). A função de codificação* $\alpha : \mathbb{Z}_2{}^7 \to \mathbb{Z}_2{}^{21}$ *é definida da seguinte forma: para cada* $w = w_0 w_1 \dots w_7 \in \mathbb{Z}_2{}^7$, *seja* $c = \alpha(w) = w_0 w_1 \dots w_7 w_0 w_1 \dots w_7 w_0 w_1 \dots w_7$, *ou seja, repetimos mais 2 vezes a mesma sequência. Depois da transmissão ou do armazenamento, suponhamos que a sequência*

$$\tau(c) = v_0 v_1 \dots v_7 v_0{}' v_1{}' \dots v_7{}' \dots v_7 v_0{}'' v_1{}'' \dots v_7{}''$$

é recebida ou lida. Vamos definir a função de decodificação $\sigma : \mathbb{Z}_2{}^{21} \to \mathbb{Z}_2{}^7$ *dessa forma:*

$$\sigma(v_0 v_1 \dots v_7 v_0{}' v_1{}' \dots v_7{}' \dots v_7 v_0{}'' v_1{}'' \dots v_7{}'') = u_0 u_1 \dots u_7$$

onde cada u_j *é igual ao valor de maior ocorrência entre* v_j, $v_j{}'$ *e* $v_j{}''$. *Nesse caso, se no máximo um bit entre* v_j, $v_j{}'$ *e* $v_j{}''$ *for diferente de* w_j, *então ainda teremos* $u_j = w_j$. *Assim, nesse caso, até um erro para cada* w_j *pode ser detectado e corrigido.*

Exemplo 3.6 Códigos Reed-Solomon *veja [58], página 100*

3.4 Códigos Cíclicos

Os códigos cíclicos são uma classe importante de códigos de bloco lineares. Estes códigos são manipulados algebricamente através de polinômios, sem a utilização de matrizes, o que facilita tanto a codificação como a decodificação.

Definição 3.10 *Um dado código de bloco linear é cíclico, se para cada um dos seus vetores de código, o i-ésimo deslocamento cíclico (LFSR) também é um vetor código.*

mensagem	*vetor código*	*polinômio código*
0000	0000000	$0 = 0.p(x)$
1000	1001100	$1 + x^3 + x^4 = 1.p(x)$
0100	0100110	$x + x^4 + x^5 = x.p(x)$
0010	0010011	$x^2 + x^5 + x^6 = x^2.p(x)$
0001	1100001	$1 + x + x^6 = x^3 + x^6 + (1 + x + x^3) = x^3 + x^6 + x^7 = x^3.p(x)$
1100	1101001	$1 + x^2 + x^4 + x^7 = x^4 + x^7 + 1 + x^2 = x^4 + x^7 + x^8 = x^4.p(x)$
0110	0111010	$x + x^2 + x^3 + x^5 = x^5 + 1 + x^2 + x + x^3 = x^5 + x^8 + x^9 = x^5.p(x)$
0011	1011001	$1 + x^2 + x^3 + x^6 = x^6 + x + x^3 + 1 + x + x^2 = x^6 + x^9 + x^{10} = x^6.p(x)$
1101	0100000	$x = 1 + x + x^3 + 1 + x + x^2 + x + x^2 + x^3 = x^7 + x^{10} + x^{11} = x^7.p(x)$
1010	0010000	$x^2 = 1 + x^2 + x + x^2 + x^3 + 1 + x + x^2 + x^3 = x^8 + x^{11} + x^{12} = x^8.p(x)$
0101	0001000	$x^3 = x + x^3 + 1 + x + x^2 + x^3 + 1 + x^2 + x^3 = x^9 + x^{12} + x^{13} = x^9.p(x)$
1110	1100000	$1 + x = 1 + x + x^2 + 1 + x^2 + x^3 + 1 + x^3 = x^{10} + x^{13} + x^{14} = x^{10}.p(x)$
0111	1110000	$1 + x + x^2 = x + x^2 + x^3 + 1 + x^3 + 0 = x^{11} + x^{14} + x^{15} = x^{11}.p(x)$
1111	0011000	$x^2 + x^3 = 1 + x^2 + x^3 + 0 + 1 = x^{12} + x^15 + x^16 = x^{12}.p(x)$
1011	1011000	$x + x^2 + x^3 = 1 + x^2 + x^3 + 1 + x = x^{13} + x^{16} + x^{17} = x^{13}.p(x)$
1001	1111000	$1 + x + x^2 + x^3 = 1 + x^3 + x + x^2 = x^{14} + x^{17} + x^{18} = x^{14}.px(x)$

Tabela 3.1: Código cíclico $CH : (7, 4)$ gerado pelo polinômio $p(x) = 1 + x^3 + x^4$

Exemplo 3.7 *$C1 = \{000, 110, 101, 011\}$ é um código cíclico. $C2 = \{000, 100, 011, 111\}$ não é um código cíclico.*

O código $CH : (7, 4)$, descrito no exemplo 2.9, também é um código cíclico.

Definição 3.11 Representação Polinomial *Uma representação polinomial de um vetor código $c = (c_0, c_1, \ldots, c_{n-1})$ é da forma:*

$$c(X) = c_0 + c_1X + c_2X^2 + \ldots + c_{n-1}X^{n-1}, c_i \in \mathbb{GF}_{2^m}$$

Exemplo 3.8 *Para esse exemplo, usaremos o corpo $\mathbb{GF}_{2^4}$ do exemplo 2.8. Na tabela 3.1, temos os vetores código desse código e como eles foram gerados a partir da representação polinomial e representação exponencial dos elementos do corpo $\mathbb{GF}_4$ (tabela 2.5).*

A representação polinomial é utilizada em todas as operações $(\oplus, \otimes, \div)$ entre os elementos de um código cíclico. Códigos cíclicos podem utilizar um esquema com o corpo $\mathbb{GF}_{2^8}$ e com os coeficientes do polinômio gerador com elementos do corpo $\mathbb{GF}_2$.

Para m grande, a determinação da representação polinomial de cada elemento de forma algébrica é inviável. O mapeamento dos elementos de $\mathbb{GF}_q$ da representação exponencial para representação polinomial pode ser realizado de forma mais genérica utilizando a idéia

de um circuito com registradores de deslocamento com realimentação linear (LFSR). Esse circuito consiste em m *flip-flops* em cascata, com realimentação do bit mais significativo.

As operações $(\oplus, \times, \div)$ entre polinômios sobre um corpo são usualmente definidas em sistemas de computadores sob o corpo $\mathbb{GF}_2$.

Definição 3.12 *Sejam dois polinômios*

$$c_1 = c_{01} + c_{11}X + \ldots + c_{n-1,1}X^{n-1}$$
$$c_2 = c_{02} + c_{12}X + \ldots + c_{n-1,2}X^{n-1},$$

com $c_{i1}, c_{i2} \in \mathbb{GF}_{2^m}$. *A adição de* c_1 *e* c_2 *é realizada pela adição módulo* 2 *dos coeficientes correspondentes ao termo* X^i*:*

$$c_1(X) \oplus c_2(X) = c_{01} \oplus c_{02} \oplus (c_{11} \oplus c_{12})X + \ldots \oplus (c_{n-1,1} \oplus c_{n-1,2})X^{n-1}$$

A multiplicação é realizada usando a multiplicação e a adição módulo 2*, como segue:*

$$c_1(X) \times c_2(X) = (c_{01} \otimes c_{02}) \oplus (c_{01} \otimes c_{12} \oplus c_{02} \otimes c_{12})X \oplus \ldots \oplus (c_{n-1,1} \oplus c_{n-1,2})X^{2(n-1)}$$

A divisão é definida para $c_2 \neq 0$ *e pela existência de dois polinômios* $q(X), r(X)$ *únicos sob o corpo* $\mathbb{GF}_{2^m}$ *tal que:*

$$c_1(X) = q(X) \otimes c_2(X) \oplus r(X)$$

Definição 3.13 Matriz Geradora Sistemática *Um código de blocos cíclico e sistemático* $C : (m, k)$ *com seu polinômio gerador* $g(X)$ *e com polinômio mensagem da forma:* $m(X) = m_0 + m_1X + \ldots + m_{k-1}X^{k-1}$ *pode ser obtido através das seguintes operações:*

1. *Calcula-se o polinômio* $X^{n-k}m(X) = m_0X^{n-k} + m_1X^{n-k+1} + \ldots + m_{k-1}X^{n-1}$

2. *Divide-se* $X^{n-k}m(X)$ *pelo polinômio gerador* $g(X)$

$$X^{n-k}m(X) = q(X)g(X) + p(X)$$

   sendo $p(X)$ *o polinômio resto da divisão com grau igual ou menor a* $n - k - 1$.

3. *Reordenando os termos*

$$p(X) + X^{n-k}m(X) = q(X)g(X)$$

Nesse formato, observamos que o polinômio $p(X) + X^{n-k}m(X)$ *é um polinômio código, pois é múltiplo de* $g(X)$*. Podemos ver que o termo* $X^{n-k}m(X)$ *é o polinômio mensagem deslocado à direita em* $n-k$ *posições e o polinômio resto da divisão é a paridade, ocupando os termos de menor grau do polinômio. Assim, o polinômio código na forma sistemática fica:*

$$c(X) = p(X) + X^{n-k}m(X) = p_0 + p_1X + \ldots + p_{n-k-1}Xn - k - 1 + m_0X^{n-k} + \ldots + m_{k-1}X^{n-1}$$

que pode ser expresso na forma de vetor código

$$c = (p_0, p_1, \ldots, p_{n-k-1}, m_0, m_1, \ldots, m_{k-1})$$

3.5 Códigos Convolacionais

P. Elias introduziu códigos convolucionais em 1955. Um código convulacional é um dispositivo com memória. Apesar de aceitar uma mensagem de entrada de tamanho fixo e produzir uma saída codificada, seus cálculos não dependem somente da entrada atual, mas também das entradas e saídas anteriores.

Um codificador para um código convolucional também aceita blocos de k bits da sequência de dados u e gera uma sequência de saída v de m *bits* chamada *codeword* ou palavra código. Cada bloco da palavra código não depende apenas dos k bits do bloco da sequência de dados correspondente, mas também de M blocos anteriores. Dizemos que o codificador tem memória de ordem M, onde M é o número de registradores de memória. Esse conjunto de blocos de k bits, o codificador das palavras código de tamanho m e de memória de ordem M é chamado de (m, k, M) código convolucional. $R = \frac{k}{m}$ é a taxa de codificação. Como o codificador tem memória, ele pode ser implementado como circuito lógico sequêncial [31].

Em um código convolucional, os $m - k$ bits redundantes, que provem à codificação a capacidade de tratar os ruídos do canal, podem ser adicionados quando $k < m$. Para uma mesma taxa de codificação R, pode-se adicionar redundância, aumentando a ordem da memória M do codificador. Como usar a memória para obter uma transmissão confiável sob um canal com ruído é o principal problema do projeto do codificador.

Códigos convolucionais podem ser usados para melhorar o desempenho da comunicação por rádio e satélites. Códigos convolucionais são utilizados nas tecnologias CDMA (*Code division multiple access*) e GSM (*Global System for Mobile Communications*) para telefones celulares, *dial-up modems*, satélites, *NASA's Deep Space Network* deep-space communications e na rede WLAN Wi-Fi IEEE 802.11.

Capítulo 4

Discussão sobre esquemas adequados de redundância de dados

Pode-se encontrar na literatura um número significativo de projetos que propõem sistemas de armazenamento distribuído, sistemas de arquivos distribuídos, ou sistemas de backup. Apesar da disso, nenhum esquema de redundância de dados tem sido amplamente aceito para esses sistemas e nenhuma regra fácil foi criada para se encontrar um esquema adequado de redundância de dados.

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. Esquemas de redundância podem implementar um conjunto destas técnicas [22].

Os autores em [2] estudaram redundância de dados em sistemas *peer-to-peer* para *backup* e propuseram um esquema hídrido para implementar redundância (replicação e codificação) de dados. Em [35], foi proposto um sistema de armazenamento de dados baseado em discos que usa dois níveis de codificação. Nesses textos, encontramos uma comparação entre alguns sistemas de armazenamento, avaliando-se algumas de suas características. Podemos observar que os tipos de esquema de redundância de dados são replicação, codificação por apagamento e híbrido, sendo a replicação, a estratégia mais utilizada nos sistemas comparados por [2] e a codificação por apagamento, para os sistemas comparados por [35].

Redundância de dados é necessária para prevenir perda de dados, mas não é suficiente. A avaliação de esquemas de redundância é muitas vezes baseada na suposição de que as réplicas falham de forma independente. Na prática, as falhas não são tão independentes, segundo [34, 43]. Esse trabalho não tratará a independência das réplicas.

Cada esquema de redundância estabelece (i) como criar os dados redundantes e (ii) como reconstruir os dados quando houver falha. Essas duas operações geram custos que

diferem de um esquema para outro. Esse trabalho comentará os mais amplamente usados esquemas de redundância: replicação e codificação por apagamento, que chamaremos de codificação.

4.1 Esquemas de redundância de dados para sistema de armazenamento

Esquemas de redundância de dados são utilizados em sistemas de armazenamento (exemplo: sistemas RAID) para prover disponibilidade, tolerância a falhas e durabilidade de dados e em sistemas de comunicação (exemplo, sistemas *peer-to-peer*) para prover uma entrega confiável e segura de dados.

A Figura 4.1 apresenta um sistema de armazenamento com um arquivo de dados 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 4.2 apresenta o mesmo sistema que a Figura 4.1, 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 4.2 também se aplica a códigos Tornado.

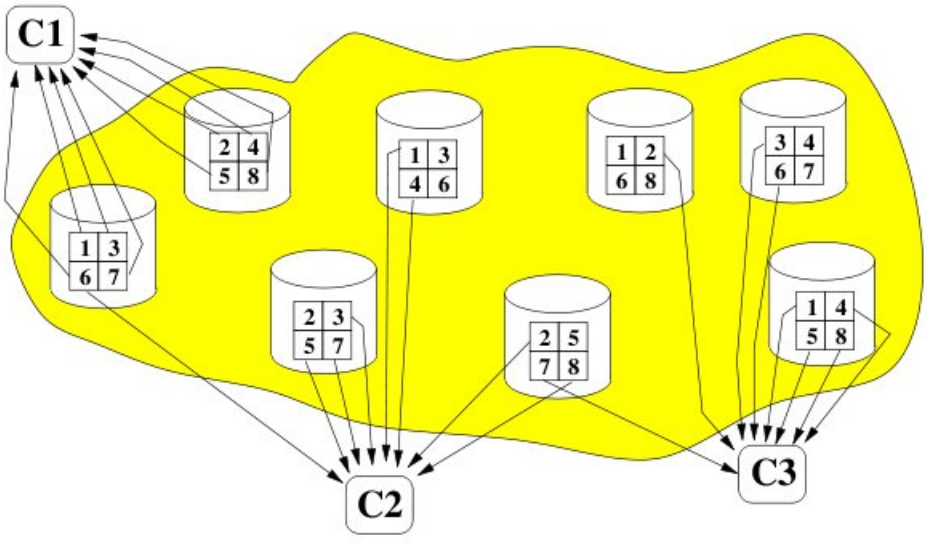

Figura 4.1: Sistema com replicação pura [23]

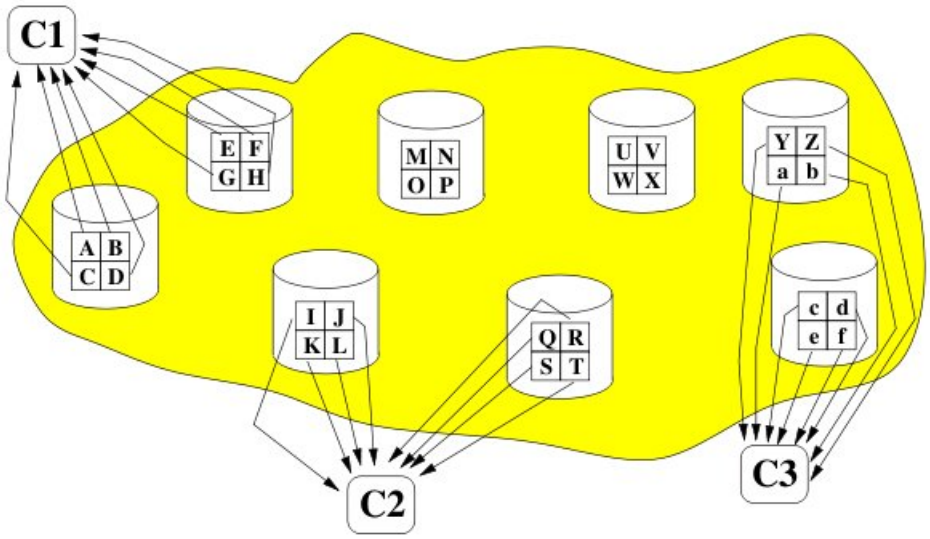

Figura 4.2: Sistema com códigos RS [23]

4.1.1 Replicação

Replicação é o esquema de redundância mais simples. A maioria dos sistemas, que utiliza redundância de dados, é baseada em replicação, mas esse esquema consume mais espaço que a codificação por apagamento, pois uma cópia completa de cada arquivo é armazenada em cada um dos servidores de dados.

A principal desvantagem da replicação é que ela requer uma grande sobrecarga 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 [42]. Por exemplo, no artigo sobre o sistema Glacier [45], os autores argumentam que o armazenamento aumenta de 11 vezes a quantidade de dados armazenados utilizando apenas replicação para conseguir 0.999999 (*six nines*) de confiabilidade num cenário com 60% de indisponibilidade dos nós.

Os autores em [50] 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.

A replicação é usada no Google File System [55] (GFS), no Hadoop Distributed File System [19] (HDFS) e no Kosmos distributed file system [61] (KFS), sistemas de arquivos distribuídos que apresentam características semelhantes. Um *cluster* do GFS ou do HDFS

ou do KFS é formado por um único servidor, master (GFS) ou namenode (HDFS) ou Meta server (KFS), que mantém os metadados e muitos servidores de dados, os chunkservers (GFS e KFS) ou os datanodes (HDFS) e é acessado por vários clientes. Os arquivos de dados são armazenados nos chunkservers (GFS e KFS) ou datanodes (HDFS) e são particionados em blocos de igual tamanho. GFS, HDFS e KFS foram projetados para aplicações que processam grande volume de dados.

Seus projetos consideram *clusters* de (*commodity hardware*), uma versão do *kernel* linux como sistema operacional para as máquinas e uma arquitetura de rede com dois níveis: vários *racks* interligados por um comutador e cada *rack* é formado por várias máquinas e seus discos, estes também interligados por um comutador. A estratégia de inserção de dados cria réplicas em *racks* distintos do *rack* onde está a 1^a réplica, assim, falhas que comprometam um *rack* não provocam a indisponibilidade de dados. Os arquivos de dados são alterados por concatenações ao invés de sobrescrever dados existentes. Após a criação, os arquivos de dados são usados apenas para leitura e esta leitura ocorre sequencialmente. O KFS permite escrever em posições randômicas nos arquivos. As APIs do cliente fornecidas pelo GFS, pelo HDFS e pelo KFS suportam operações de criação, leitura, escrita, remoção de arquivos, mas não implementam a interface POSIX.

O GFS está disponível para linux sob uma licença de software proprietário. O HDFS [1] e o KFS [2] estão disponíveis para linux sob uma licença Apache.

O Farsite, que utiliza apenas replicação, é um sistema de arquivos distribuídos, particionados em namespaces, explorando os desktops presentes dentro da Microsoft, sem servidor mestre, disponível para Windows sob uma licença de software proprietário. A escolha da replicação foi, pelos autores, considerada uma opção mais simples para disponibilidade, já que a codificação poderia significar latência adicional nas leituras dos arquivos. Ainda segundo os autores, estudos com experimentos já mostraram que a codificação pode apresentar um bom desempenho e seria possível, então, alterar no futuro o esquema de redundância do Farsite.

Big Table (construído sob o GFS) e Dynamo (construído para Amazon.com) são dois sistemas de armazenamento que gravam e recuperaram dados através de uma chave e executam em um *pool* compartilhado de máquinas, utilizam apenas replicação.

Ceph [9] é um sistema de arquivos *open source* que possui três principais componentes: um *cluster* de servidores de metadados (que gerencia o namespace, nomes de arquivos e diretórios), um *cluster* de OSDs (dispositivos de armazenamento de objetos) que armazenam dados e metadados e os clientes que utilizam uma interface do sistema de arquivos. O Ceph agrupa dados em PGs (grupos de colocação) e usa uma função *hash* para distribuir os PGs nos OSDs, cujo algoritmo CRUSH é $O(logn)$ e usa uma árvore-B

[1] http://hadoop.apache.org/
[2] http://code.google.com/p/kosmosfs/

para indexar os PGs. Existe um módulo em desenvolvimento que permite usar o Ceph como armazenamento para uma instância do Hadoop. O Ceph utiliza apenas replicação, implementa parcialmente a interface POSIX e está disponível para linux sob LGPL [3].

Lustre [4] tem na sua arquitetura os metadata server (disponibiliza os metadados para clientes), o metadata target (um por sistema de arquivos, armazena os metadados), object storage servers (armazena os dados), object storage target (armazena os objetos que contém os arquivos de dados) e clientes.

Moosefs [5] também foi projetado com uma arquitetura que se assemelha a do GFS, HDFS e KFS: master server (que armazena os metadados), chunk servers (que armazenam os dados), metalogger server (podem substituir algumas funções do master server, se ele falhar) e clientes (que solicitam dados e se comunicam com o master server e o chunk servers.

Ambos, Lustre e Moosefs, usam replicação, implementam a interface POSIX e estão disponíveis para linux sob uma licença GPL.

Com exceção do Farsite, os sistemas de armazenamento apresentados consideram *clusters* de (*commodity hardware*) e uma versão do *kernel* linux como sistema operacional para as máquinas.

4.1.2 Codificação por Apagamento

Uma vantagem de codificação por apagamento é 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 [14]: para um mesmo espaço de armazenamento, o tempo médio entre falhas (*mean time to failure*) é maior.

HDFS, Total Recall e OceanStore usam codificação para reduzir o tamanho do armazenamento de dados e todos os outros sistemas usam codificação para prover disponibilidade e confiabilidade.

Os sistemas que foram comparados na tabela 4.1, estão disponíveis para uma versão de sistema operacional linux ou unix. Além da codificação, todos eles implementam replicação.

Vamos avaliar algumas das métricas utilizadas em literatura para comparar redundância de dados em sistemas de armazenamento: sobrecarga de armazenamento, disponibilidade dos nós, corrupção de um dado e operações de criação, leitura, atualização consistente e remoção de dados redundantes. Inicialmente vamos definir alguns conceitos adaptados de [2, 21, 6, 63] para esses dois esquemas de redundância.

[3] http://ceph.newdream.net/
[4] git://git.lustre.org/prime/lustre.git
[5] http://www.moosefs.org/

sistema	**codificação**	**arquitetura do sistema**	**licença**
HDFS [19]	RAID, RS, ...	shared-disk file system for cluster	Apache
Tahoe-LAFS [7] [6]	RAID	peer-to-peer filesystem	GPL2
Pergamum [35]	XOR parity, RS	disk-based archival storage	*
Potshards [36]	RAID e RS	disk-based archival storage	*
RobuStore [28]	Luby Transform (LT)	disk-based archival storage	*
Glacier [45]	RS com matrizes Cauchy	peer-to-peer storage system	*
Total Recall [39]	Maymounkov's online codes	peer-to-peer storage system	*
FAB [57]	RS	distributed disk array	*
GPFS [53]	RAID	shared-disk file system for cluster	*
Oceanstore [4] [7]	Tornado e RS	desktops e notebooks conectados a servidores geograficamente distribuídos	BSD
xFS [10] [8]	RAID	serverless network file system	GPL
Swift [15]	RAID	desktops sob unix conectados a uma intranet	*

Tabela 4.1: Comparação de codificação entre sistemas de armazenamento de grande volume de dados que utilizam *commodity hardware*

onde:
* = não disponível

4.2 Caracterização da replicação e da codificação

A confiabilidade de um esquema de redundância é medida pelo número de falhas simultâneas que ele pode tolerar sem comprometer a capacidade de reconstruir os dados originais. Assim esta propriedade pode ser expressada como a **probabilidade de perda de dados, dado que ocorreram l falhas**, $P(l)$.

Para avaliar o armazenamento, vamos definir **fator de redundância** B, uma razão entre tamanho dos dados originais mais a redundância $|dado + red|$ e o tamanho dos dados originais $|dado|$ e também vamos definir grau de reparação.

O **grau de reparação** mede o que deve ser feito após parte da redundância ser perdida. Para isso, é feita uma leitura de dados disponíveis para produzir novos. O custo dessa leitura em um sistema de armazenamento distribuído inclue volume do tráfego da rede, política de reparação, algoritmo de coordenação. Nesse estudo vamos apenas avaliar a contribuição do esquema de redundância: a quantidade de dados a serem lidos para que dados novos sejam criados para reparar o dado corrompido, definido por d.

Definimos p como a probabilidade de um nó estar disponível e $q = 1 - p$ como a probabilidade de um nó não estar disponível. Nesse estudo, assumimos que elas são iguais para todos os nós.

Vamos definir as operações de acesso a dados redundantes como uma tupla (r, w, a), onde r é o número mínimo de nós disponíveis que armazenam dados redundantes, w é o número mínimo de nós disponíveis que armazenam dados redundantes que devem ser acessados para armazenar novos valores e a é número de nós (provavelmente outros) que devem estar disponíveis para completar a operação.

4.2.1 Replicação

Para as definições, n é o número de réplicas dos dados.

número de falhas suportadas $l = n - 1$

probabilidade de perda de dados, dado que ocorreram l falhas

$$P(l) = \begin{cases} 0, & \text{se } l < n \\ 1, & \text{se } l = n \end{cases}$$

fator de redundância

$$B = \frac{|dado + red|}{|dado|} = n$$

grau de reparação

$$d = 1$$

acesso	**tupla** (r, w, a)
criar	$(0, 0, n)$
apenas leitura	$(1, 0, 0)$
atualização consistente	$(1, n, 0)$
apagar	$(0, n, 0)$

Tabela 4.2: Operações sobre dados redundantes na replicação

disponibilidade de um dado

$$1 - q^n$$

corrupção de um dado

$$e = q^n$$

Comparando as tuplas (1, 0, 0) e (1, n, 0) da tabela 6.1, podemos observar que a disponibilidade da operação "atualização consistente"é menor que a disponibilidade da operação "apenas leitura", quando o número de réplicas $n > 1$ é aplicado.

4.2.2 Codificação por Apagamento

Na Codificação (m, k), k é o número de blocos originais do dado, m é o número de blocos codificados e $m - k$ é o número de blocos adicionados pela codificação.

número de falhas suportadas $l = m - k$

probabilidade de perda de dados, dado que ocorreram l **falhas**

$$P(l) = \begin{cases} 0, & \text{se } l <= m - k \\ 1, & \text{se } l > m - k \end{cases}$$

fator de redundância

$$B = \frac{m}{k}$$

grau de reparação

$$d = k$$

disponibilidade de um dado

$$B(k, m, p) = \sum_{i=k}^{m} \binom{m}{i} p^i q^{m-i}$$

corrupção de um dado

$$e = q^{m-k}$$

acesso	**tupla** (r, w, a)
criar	$(0, 0, n)$
apenas leitura	$(k, 0, 0)$
atualização consistente	$(k, m-k+1, k-1)$
apagar	$(0, m-k+1, 0)$

Tabela 4.3: Operações sobre dados redundantes na codificação por apagamento

Por outro lado, comparando-se as tuplas $(k, 0, 0)$ e $(k, m-k+1, k-1)$ da tabela 4.3, podemos observar que existe um caso no qual as operações "apenas leitura"e "atualização consistente"podem ter a mesma disponibilidade. Isto ocorre quando $m = 2k-1$ (assumindo-se também que número total de nós disponíveis é maior ou igual a m), obtendo-se a tupla $(k, k, k-1)$.

4.3 Sobrecarga de armazenamento

Em [14, 50], os autores afirmam que a codificação obtem o mesmo nível de disponibilidade como a replicação, usando muito menos espaço de armazenamento.

Em [39], concluiu-se que se a disponibilidade do nó for 0.5, então isso requer 10 cópias de cada arquivo para garantir a disponibilidade de 0.999 dos arquivos.

No esquema da replicação, para um objetivo de probabilidade de indisponibilidade de um dado $e = 0.0001$, dado uma probabilidade de um nó estar indisponível $q = 0.05$, obtemos o valor de $n = 3$ réplicas.

$$\begin{array}{c} e = q^n \\ n = \frac{\log e}{\log q} = \frac{\log 0.0001}{\log 0.05} = 3.074487147 \\ n \approx 3 \end{array}$$

No esquema da codificação, para o mesmo objetivo, para as mesmas probabilidades e e q, obtemos um valor de 4 blocos de paridade para uma codificação $(2k-1, k)$.

$$\begin{array}{c} e = q^{m-k} \\ m - k = \frac{\log e}{\log q} \end{array}$$

Substituindo m por $2k-1$ e $\frac{\log e}{\log q}$ por 3.074487147,

$$\begin{array}{c} k - 1 = 3.074487147 \\ k \approx 4 \end{array}$$

Para uma mesma probabilidade de indisponibilidade de um dado $e = 0.0001$ e uma mesma probabilidade de um nó estar indisponível $q = 0.05$, obtemos o valor de $n = 3$

réplicas para a replicação e um valor de $k = 4$ blocos iniciais para a codificação (2k-1,k). O fator de redundância é $B = 3$ para a replicação e $B = \frac{m}{k} = \frac{2k-1}{k} = \frac{7}{4} = 1.75$ para a codificação.

Na figura 4.3, podemos observar o gráfico das funções $y = 2x$, $y = 3x$ e $y = \frac{2x-1}{x}$, para $x > 0$, representando a sobrecarga de armazenamento na replicação $2n$ e na $3n$ e na codificação $(2k-1, k)$, respectivamente. O eixo x representa o tamanho de um dado e o eixo y, a sobrecarga de armazenamento desse dado.

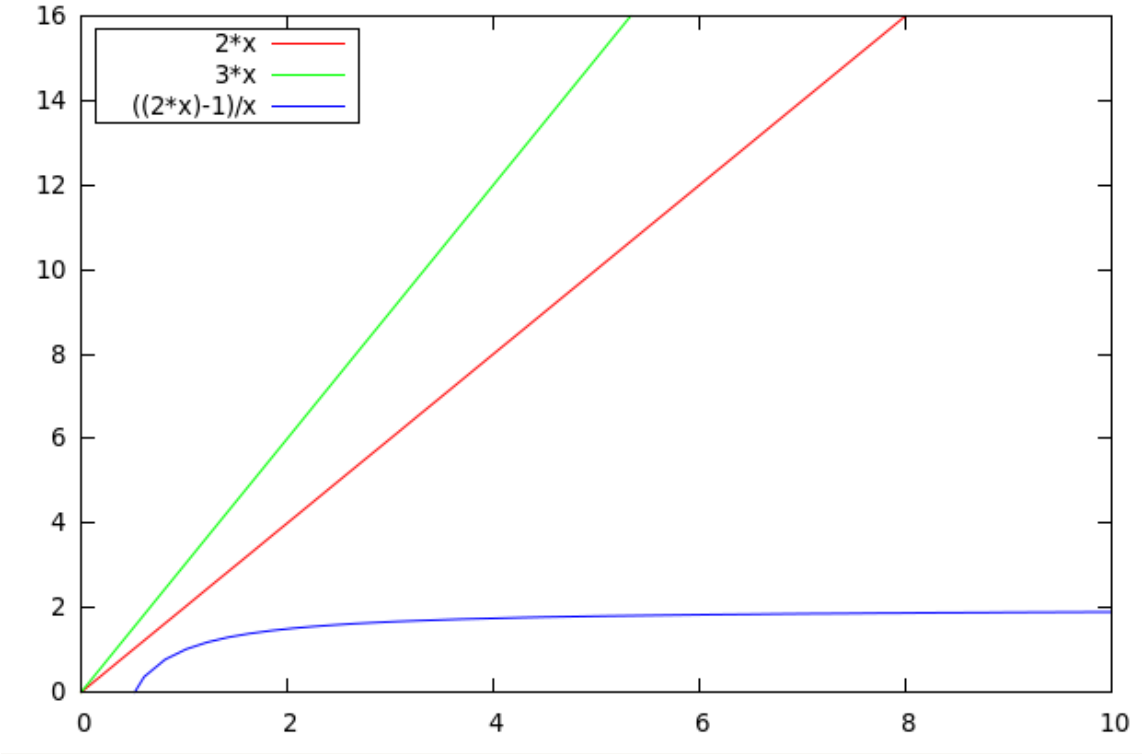

Figura 4.3: Sobrecarga de armazenamento para replicação $3n$, $2n$ e codificação $(2k-1, k)$ representadas, respectivamente, pelas funções $y = 3x$, $y = 2x$ e $y = \frac{2x-1}{x}$, para $x > 0$

Neste estudo, concluímos que a codificação por apagamento acarreta uma sobrecarga de armazenamento menor que a replicação para o mesmo número de falhas toleradas.

Nas máquinas do Facebook, a codificação RS é (13, 10) com sobrecarga de ($B = 1.3$) e tolera até 3 falhas por *stripe*. A codificação XOR (RAID-5) é (11, 10) com sobrecarga de ($B = 1.1$). Outro exemplo é a codificação RS(255, 223), utilizada pela DSN [66], que é padrão CCSDS [9] e tolera 32 falhas com sobrecarga de $B = 1.1435$. Uma codificação BCH(63, 56), também padrão CCSDS, pode tolerar até 7 falhas, dependendo do polinômio gerador da síndrome [38] .

4.4 Disponibilidade dos nós

4.5 Leitura ou Atualização dos dados redundantes

Em [21], os autores concluem que:

- o acesso somente de leitura pode ser suportado tanto por replicação de dados simples como por codificação
- para privilegiar atualização consistente, uma codificação de alta disponibilidade é necessária que se caracteriza por fracionamento do original dados em pedaços k e adicionando exatamente $k - 1$ pedaços
- se ler e a disponibilidade de atualização consistente são de igual importância, isso requer codificação $(2k - 1, k)$

Os autores também concluíram que usar apenas a replicação tem sentido apenas em poucos casos.

Concluímos que, apenas avaliando a contribuição do esquema de redundância, a codificação por apagamento

Podemos também concluir que a replicação é um caso onde $k = 1$ e $l = m-k = n-1$, portanto $m = n$.

[9] http://public.ccsds.org/

Capítulo 5

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

Em 2004, um artigo [54], 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 de forma não estruturada, com esquema dinâmico e integridade baixa. Uma lista de aplicações e organizações que usam o Hadoop pode ser encontrada em [68].

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

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

5.1 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 [62].

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 HDFS/MapReduce que podem ser escritos em várias linguagens como Java, Ruby, Python e C++.

5.2 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 5.2 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 5.1. Cada *rack* por sua vez é formado por vários nós (máquinas) e seus discos, estes também interligados por um comutador.

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

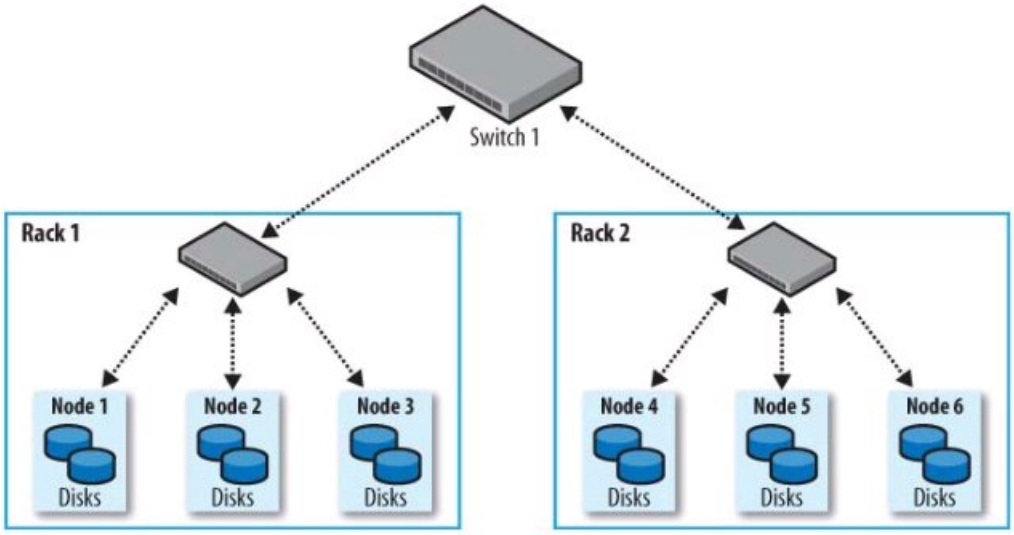

Figura 5.1: Arquitetura de rede em dois níveis para um cluster Hadoop [19]

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 5.3 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 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 [19].

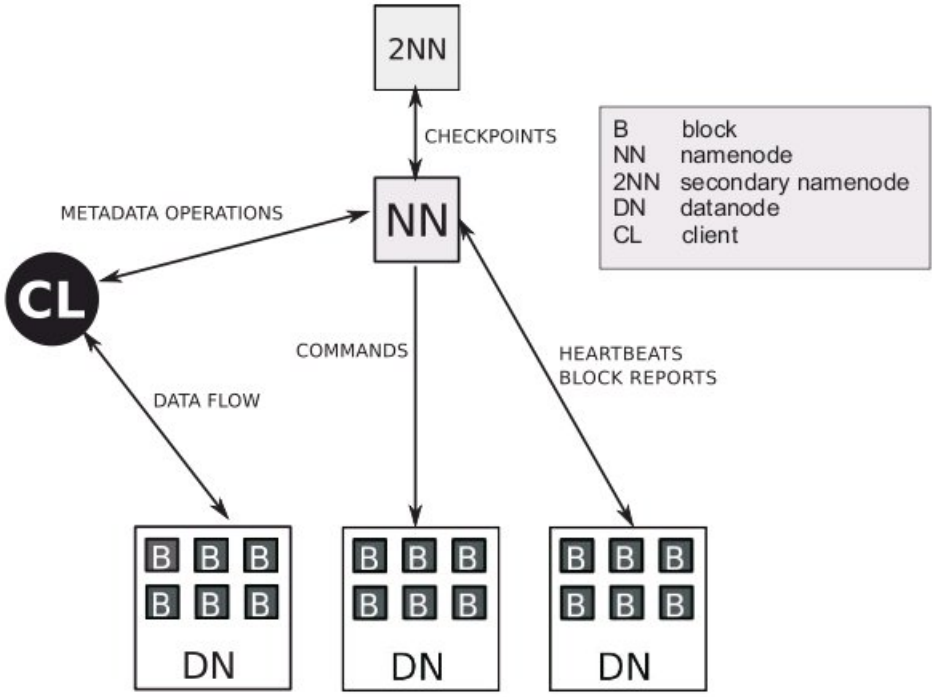

Figura 5.2: Arquitetura do HDFS [52]

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.

5.3 Codificação por Apagamento

Existe uma nova característica proposta em 2009 para implementação de uma camada de codificação por apagamento no Hadoop utilizando XOR (RAID-5) [18] e uma mais recente utilizando códigos RS [20]. A inclusão da codificação por apagamento foi proposta com o objetivo de reduzir o tamanho do armazenamento do HDFS.

A versão atual do Hadoop não utiliza apenas a técnica de replicação [62] para obter disponibilidade e confiabilidade de dados. Ela pode ser configurada para usar também codicação XOR (RAID-5) e RS.

No HDFS, a codificação RS *default* é (8, 5) e a XOR *default* é (6, 5). Logo, toleram até

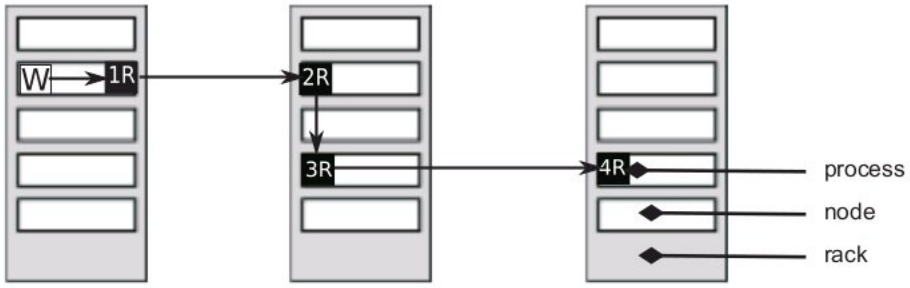

Figura 5.3: Arquitetura do HDFS - Datanodes e Blocos [62]

$l = 3$ e até $l = 1$ falhas, respectivamente, por *stripe* de 5 blocos ($k = 5$) com sobrecarga de $B = 1.6$ e de $B = 1.2$, respectivamente. Ambas foram projetadas sob o corpo $GF(2^8)$. O polinômio primitivo dessa codificação RS é $x^8 + x^4 + x^3 + x^2 + 1$.

Na codificação XOR, é possível configurar o número de blocos por *stripe*. Já na codificação RS, tanto o número de blocos por *stripe* e como o número de blocos de paridade são configuráveis.

5.3.1 Algoritmos da Camada RAID

Exemplo do algoritmo de codificação

O tamanho da *stripe* é 5 blocos e existe um arquivo */a/arquivo.txt* com exatamente 5 blocos. Nesse caso, o algoritmo de codificação da camada RAID faz o seguinte:

```
bloco[0] = primeiro bloco
bloco[1] = segundo bloco
bloco[2] = terceiro bloco
bloco[3] = quarto bloco
bloco[4] = quinto bloco

bloco_paridade = iniciado com 0 em todos os bytes

para i de 0 até número de bytes em um bloco:
   para j de 0 até 4:
      bloco_paridade = bloco_paridade xor bloco[j][i]

para i de 0 até 4:
```

```
   escreva bloco_paridade no arquivo /raid/a/arquivo.txt
```

5.3.2 Algoritmos da Camada RS

Exemplo do algoritmo de codificação

O tamanho da *stripe* é 5 blocos, são 3 blocos de paridade, o polinômio primitivo é $x^8 + x^4 + x^3 + x^2 + 1$ e existe um arquivo $/a/arquivo.txt$ com exatamente 5 blocos. Nesse caso, o algoritmo de codificação da camada RS faz o seguinte:

```
// polinômio primitivo g representado por 285
g = x^8 + x^4 + x^2 + x + 1
m = 8
k = 5
i = 0
L = m-k
j = 0

bloco[0] = primeiro bloco
bloco[1] = segundo bloco
bloco[2] = terceiro bloco
bloco[3] = quarto bloco
bloco[4] = quinto bloco

enquanto j >= L faça
   // para GF = 2
   // x^0 representado por 1; x^1 por 2; x^2 por 4; x^3 por 8;
   // x^4 por 16; x^5 por 32; x^6 por 64; x^7 por 128;
   a = x^j
   i = 0
   bloco_paridade[j] = iniciado com 0 em todos os bytes
   enquanto i >= k faça
      r = novo bloco
      resto(g, a, bloco[i], r)
      bloco_paridade[j] = bloco_paridade[j] xor r
      i++
   j++

para j de 0 até L
   escreva bloco_paridade[j] no arquivo /raidrs/a/arquivo.txt
```

```
resto (g, a , bloco, r)
   para i de 0 até número de bytes em um bloco:
      r[i] = resto(g, a, bloco[i])

resto (g, a, b)
retorna resto = (byte) rem (g(b), a(b))
```

Capítulo 6

Implementação das Codificações

Uma boa escolha de codificação para um canal depende da sobrecarga de armazenamento, do tamanho do bloco, da complexidade dos algoritmos de codificação e de decodificação e se uma eventual não detecção de erros é aceitável. Códigos LDPC e Tornado são versáteis para serem projetados para muitas opções de tamanho do bloco e de sobrecarga de armazenamento e segundo muitos estudos, códigos LDPC são a codificação que mais se aproxima do limite de Shannon (quantidade máxima de dados transmitidos em um canal em bytes por segundo).

6.1 Codificação Tornado

A idéia básica deste algoritmo da codificação Tornado está descrita em [42, 33].

O cálculo do checksum de 64 bits foi feito para cada 100 bytes. Para um bloco de 64MB, o checksum é de 5.12MB.

A sobrecarga de armazenamento de checksum é $\frac{864}{800} = 1.08$. A sobrecarga da paridade é $\frac{2n}{n} = 2$. Portanto, a sobrecarga total de armazenamento é $1.08 * 2 = 2.16$.

O número de falhas suportadas é $l = \textit{numero de blocos da stripe} = s$.

A operação mais custosa dos algoritmos é o cálculo do XOR entre 2 blocos de 64M. São $\frac{s}{2}$ operações, no caso de s par e $\frac{n}{2} - 1$ operações, no caso de s ímpar. Nesse caso, podemos afirmar que os algoritmos são lineares no tamanho da entrada, ou seja, $O(s)$.

O grafo da codificação para uma *stripe* de tamanho 10 blocos é:

```
1 0 0 0 0 0 0 0 0 0
1 1 0 0 0 0 0 0 0 0
0 1 1 0 0 0 0 0 0 0
0 0 1 1 0 0 0 0 0 0
0 0 0 1 1 0 0 0 0 0
```

```
0 0 0 0 1 1 0 0 0 0
0 0 0 0 0 1 1 0 0 0
0 0 0 0 0 0 1 1 0 0
0 0 0 0 0 0 0 1 1 0
0 0 0 0 0 0 0 0 1 1
```

6.1.1 Algoritmo de Codificação

Apresentamos o algoritmo de codificação para uma *stripe* de tamanho s blocos e que gera s blocos de paridade.

```
// Ler o arquivo /a/arquivo.txt
para i de 0 até s-1:
   bloco[i] = i-ésimo bloco da stripe

para i de 0 até s-1:
   se i = 0
   então
      bloco_paridade[i] = bloco[i]
   senão
      bloco_paridade[i] = bloco[i-1] xor bloco[i]
   i++

para i de 0 até s-1:
   escreva bloco_paridade[i] no arquivo /tor/a/arquivo.txt
```

6.1.2 Algoritmo de Decodificação

O algoritmo de decodificação faz o seguinte:

```
// Ler  o arquivo  /tor/a/arquivo.txt
para i de 0 até s-1:
   bloco_paridade[i] = i-ésimo bloco de paridade da stripe

para i de 0 até s-1:
   se i = 0
   então
      bloco[i] = bloco_paridade[i]
   senão
```

```
      bloco$[i] = bloco_paridade[i] xor bloco[i-1]
   i++
```

6.2 Codificação Simples Turbo-*Like*

Esse algoritmo foi baseado nos estudos de [29, 8] e é muito simples de entender.

A sobrecarga total de armazenamento é $\frac{2n}{n} = 2$.

O número de falhas suportadas é $l = numero\ de\ blocos\ da\ stripe = s$.

A operação mais custosa dos algoritmos é o cálculo do XOR entre 2 blocos de 64M. São s operações. Nesse caso, podemos afirmar que os algoritmos são lineares no tamanho da entrada, ou seja, $O(s)$.

O grafo da codificação para uma *stripe* de tamanho 10 blocos na sequência $7, 1, 4, 8, 2, 5, 9, 3, 0, 6$:

```
0 0 0 0 0 0 0 1 0 0
0 1 0 0 0 0 0 1 0 0
0 1 0 0 1 0 0 0 0 0
0 0 0 0 1 0 0 0 1 0
0 0 1 0 0 0 0 0 1 0
0 0 1 0 0 1 0 0 0 0
0 0 0 0 0 1 0 0 0 1
0 0 0 1 0 0 0 0 0 1
1 0 0 1 0 0 0 0 0 0
1 0 0 0 0 0 1 0 0 0
```

6.2.1 Algoritmo de Codificação

Apresentamos o algoritmo de codificação para uma *stripe* de tamanho s blocos e que gera s blocos de paridade.

```
// Ler o arquivo /a/arquivo.txt
para i de 0 até s-1:
   bloco[i] = i-ésimo bloco da stripe

Gerar uma permutação dos índices dos blocos da stripe: p = [b_0,
b_1, b_2, b_3, ..., b_{s-2}, b_{s-1}]

para i de 0 até s-1:
   se i = 0
   então
```

```
      bloco_paridade[i] = bloco[p[i]]
   senão
      bloco_paridade[i] = bloco_paridade[i-1] xor bloco[p[i]]
   i++

para i de 0 até s-1:
   escreva bloco_paridade[i] no arquivo /turbo/a/arquivo.txt
```

6.2.2 Algoritmo de Decodificação

O algoritmo de decodificação faz o seguinte:

```
Ler a permutação do índices dos blocos da stripe: p =  [b_0,
b_1, b_2, b_3, ..., b_{s-2}, b_{s-1}]

// Ler  o arquivo  /turbo/a/arquivo.txt
para i de 0 até s-1:
   bloco_paridade[i] = i-ésimo bloco de paridade da stripe

para i de 0 até s-1:
   se i = 0
   então
      bloco[p[i]] = bloco_paridade[i]
   senão
      bloco[p[i]] = bloco_paridade[i-1] xor bloco_paridade[i]
   i++
```

6.3 Comparação entre as Codificações

Codificação	**Número de falhas suportadas**	**Espaço de armazenamento**	**Grau de reparacao**	**Corrupção de um dado**	**Complexidade dos algoritmos**
Replicação 2X	$l = 2$	$B = 2$	$d = 1$	$e = q^2$	$O(1)$
RAID (6, 5)	$l = 1$	$B = 1.2$	$d = 5$	$e = q$	$O(5 \log 6)$
RS (8, 5)	$l = 3$	$B = 1.6$	$d = 5$	$e = q^3$	$O(5(3 \log 8))$
Tornado (10, 5)	$l = 5$	$B = 2$	$d = 5$	$e = q^5$	$O(10)$
Turbo-*Like* (10, 5)	$l = 5$	$B = 2$	$d = 5$	$e = q^5$	$O(10)$
Replicação nX	$l = n$	$B = n$	$d = 1$	$e = q^{n-1}$	$O(n)$
RS $(2k, k) ou (2k - 1, k)$	$l = k$	$B = 2$	$d = k$	$e = q^k$	$O(k^2 \log 2k)$[37]
Tornado $(2k, k) ou (2k - 1, k)$	$l = k$	$B = 2$	$d = k$	$e = q^k$	$O(2k)$[37]
Turbo-*Like* $(2k, k) ou (2k - 1, k)$	$l = k$	$B = 2$	$d = k$	$e = q^k$	$O(2k)$

Tabela 6.1: Comparação entre as codificações implementadas (nem todas ainda!) no HDFS

Capítulo 7

Conclusões

Referências Bibliográficas

[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] Duminoco, A. *Data Redundancy and Maintenance for Peer-to-Peer File Backup Systems*. PhD thesis, École Doctorale d´Informatique, Télécommunications et Électronique de Paris, Paris, France, October 2009.

[3] Costello, D. J. Jr. Hagenauer, J. Imai, J. H. Wicker, S. B. Applications of error-control coding. In *Proceedings of the IEEE Transactions on Information Theory*, pages 2531–2560,, Washington, DC, USA, 1998. IEEE Computer Society.

[4] 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. In *ASPLOS '00: Proceedings of the ninth international conference on Architectural support for programming languages and operating systems*, ASPLOS-IX, pages 190–201, New York, NY, USA, 2000. ACM.

[5] 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.

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

[7] Wilcox-O'Hearn, Z. Warner, B. Tahoe: the least-authority filesystem. In *Proceedings of the 4th ACM international workshop on Storage security and survivability*, StorageSS '08, pages 21–26, New York, NY, USA, 2008. ACM.

[8] MacKay, D. J. C. *Information Theory, Inference, and Learning Algorithms*, chapter 49, pages 582–587. Cambridge University Press, Cambridge, England, 2003.

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

[10] Anderson, T. Dahlin, M. Neefe, J. Roselli, D. Wang, R. Patterson, D. Serverless network file systems. Technical report, University of California at Berkeley, Berkeley, CA, USA, 1998.

[11] Luby, M. Mitzenmacher, M. Shokrollah, A. Spielman, D. Analysis of low density codes and improved designs using irregular graphs. In *STOC '98: Proceedings of the thirtieth annual ACM symposium on Theory of computing*, pages 249–258, New York, NY, USA, 1998. ACM.

[12] Ritter, D. Um estudo sobre códigos corretores de erros em espaços sobre posets. Master's thesis, Instituto de Matemática, Estatística e Computação Científica - Departamento de Matemática, Campinas, Brasil, fevereiro 2009.

[13] Vadala, D. *Managing RAID on Linux*, chapter 1, pages 1–10. O'Reilly, Sebastopol, CA, USA, 2002.

[14] 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.

[15] Cabrera, L. Long, D. D. E. Swift: Using distributed disk striping to provide high i/o data rates. Technical report, University of California at Santa Cruz, Santa Cruz, CA, USA, 1991.

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

[17] 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.

[18] 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.

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

[20] 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.

[21] Chiola, G. An empirical study of data redundancy for high availability in large overlay networks. In *Proceedings of the Second International Workshop on Hot Topics in Peer-to-Peer Systems*, pages 43–51, Washington, DC, USA, 2005. IEEE Computer Society.

[22] 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.

[23] 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.

[24] Anvin, P. H. The mathematics of raid-6, 2009. URL=http://kernel.org/pub/linux/kernel/people/hpa/raid6.pdf. Acessado em 01 de agosto de 2010.

[25] Connell, E. H. *Elements of Abstract and Linear Algebra*, chapter 2, 3, pages 19–52. University of Miami, Coral Gables, Florida, USA, 2004.

[26] Patterson, D. A. Gibson, G. Katz, R. H. A case for redundant arrays of inexpensive disks (raid). In *SIGMOD '88: Proceedings of the 1988 ACM SIGMOD International Conference on Management of Data*, pages 109–116, New York, NY, USA, 1988. ACM.

[27] Rosen, K. H. *Discrete Mathematics and its Applications*, chapter 2, 3, pages 119–299. McGraw-Hill, New York, NY, USA, 2003.

[28] Xia, H. *Robustore: a distributed storage architecture with robust and high performance*. PhD thesis, University of California at San Diego, La Jolla, CA, USA, 2006. AAI3225997.

[29] Divsalar, D. Jin, H. McEliece, R. J. Coding theorems for "turbo-like"codes. In *Proceedings of the 36th Allerton Conference on Communication, Control, and Computing*, pages 201–210, 1998.

[30] Ferreira, A. J. material de apoio - códigos detectores e correctores de erros. URL=http://www.cc.isel.pt/paginaspessoais/ArturFerreira.aspx. Acessado em 24 de julho de 2010.

[31] Lin, S. Costello, D. J. J. *Error Control Coding: Fundamentals and Applications*, chapter 1, 2, pages 1–50. Prentice-Hall Press, Englewood Cliffs, New Jersey, USA, 1983.

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

[33] Stoten, J. Digital fountains. URL=http://www.jonathanstoten.com/fountains.html. Acessado em 10 de julho de 2011.

[34] Weatherspoon, H. Moscovitz, T. Kubiatowicz, J. Introspective failure analysis: Avoiding correlated failures in peer-to-peer systems. *21st IEEE Symposium on Reliable Distributed Systems*, 0:362–367, 2002.

[35] Storer, M. W. Greenan, K. M. Miller, E. L. Voruganti, K. Pergamum: replacing tape with energy efficient, reliable, disk-based archival storage. In *FAST'08: Proceedings of the 6th USENIX Conference on File and Storage Technologies*, FAST'08, pages 1:1–1:16, Berkeley, CA, USA, 2008. USENIX Association.

[36] Storer, M. W. Greenan, K. M. Miller, E. L. Voruganti, K. Potshards: a secure, recoverable, long-term archival storage system. *Trans. Storage*, 5:5:1–5:35, June 2009.

[37] M. Luby. Lt codes. In *Proceedings of the 43rd Symposium on Foundations of Computer Science*, FOCS '02, pages 271–, Washington, DC, USA, 2002. IEEE Computer Society.

[38] 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.

[39] Bhagwan, R. Tati, K. Cheng, Y. Savage, S. Voelker, G. M. Total recall: system support for automated availability management. In *Proceedings of the 1st conference on Symposium on Networked Systems Design and Implementation - Volume 1*, pages 25–25, Berkeley, CA, USA, 2004. USENIX Association.

[40] Mitzenmacher, M. Digital fountains: A survey and look forward. In *ITW '04: Proceedings of the IEEE Information Theory Workshop*, pages 271–276, Piscataway, NJ, USA, 2004. IEEE Press.

[41] Weber, J. M. *Bounds and Constructions for Binary Block Codes Correcting Asymmetric or Unidirectional Errors*. PhD thesis, Delft University of Technology, Delft, South Holland, Netherlands, 1985.

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

[43] Baker, M. Shah, M. Rosenthal, D. S. H. Roussopoulos, M. Maniatis, P. Giuli, TJ Bungale, P. A fresh look at the reliability of long-term digital storage. In *Proceedings of the 1st ACM SIGOPS/EuroSys European Conference on Computer Systems*, EuroSys '06, pages 221–234, New York, NY, USA, 2006. ACM.

[44] Berlekamp, E. R. Peile, R. E. Pope, S. P. The application of error control to communications. *IEEE Communications Magazine*, 25(4):44–57, April 1987.

[45] 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, Boston, MA, USA, 2005. USENIX Association.

[46] Wikipedia project. Data striping. URL=http://en.wikipedia.org/wiki/Data_striping. Acessado em 01 de agosto de 2010.

[47] Wikipedia project. Low-density parity-check code. URL=http://en.wikipedia.org/wiki/Low-density_parity-check_code. Acessado em 04 de maio de 2010.

[48] Wikipedia project. Wimax. URL=http://en.wikipedia.org/wiki/WiMAX. Acessado em 04 de maio de 2010.

[49] 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.

[50] 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.

[51] Gallager, G. R. *Low-Density Parity-Check Codes*. revised doctoral dissertation, Department of Electrical Engineering, MIT, Cambridge, Massachusetts, 1963.

[52] 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.

[53] Schmuck, F. Haskin, R. Gpfs: A shared-disk file system for large computing clusters. In *Proceedings of the 1st USENIX Conference on File and Storage Technologies*, FAST '02, Berkeley, CA, USA, 2002. USENIX Association.

[54] 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.

[55] Ghemawat, S. Gobioff, H. Leung, S. The google file system. In *Proceedings of the nineteenth ACM symposium on Operating systems principles*, SOSP '03, pages 29–43, New York, NY, USA, 2003. ACM.

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

[57] Saito, Y. Frølund, S. and A. Spence, S. Veitch, A. Merchant. Fab: building distributed enterprise disk arrays from commodity components. In *Proceedings of the 11th international conference on Architectural support for programming languages and operating systems*, ASPLOS-XI, pages 48–58, New York, NY, USA, 2004. ACM.

[58] Hefez, A. Villela, M. L. T. *Códigos Corretores de Erros*, chapter 1, 2, 3, 4, 5, 6, pages 1–125. IMPA, Rio de Janeiro, RJ, Brasil, 2008.

[59] 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.

[60] Klove, T. *Codes for Error Detection*, chapter 1, pages 1–32. World Scientific Publishing Company, Englewood Cliffs, New Jersey, USA, 2007.

[61] Schürmann, T. The kosmos distributed fs, 2008. URL=http://www.linux-magazine.com/Issues/2008/90/Kosmos-FS. Acessado em 30 de junho de 2011.

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

[63] Williams, C. Huibonhoa, P. Holliday, J. Hospodor, A. Schwarz, T. Redundancy management for p2p storage. In *Proceedings of the Seventh IEEE International Symposium on Cluster Computing and the Grid*, CCGRID '07, pages 15–22, Washington, DC, USA, 2007. IEEE Computer Society.

[64] Hamming, R. W. Error detecting and error correcting codes. *The Bell System Technical Journal*, 29(2):147–160, 1950.

[65] Kurose, J. F. Ross, K. W. *Redes de Computadores e a Internet: uma abordagem top-down*, chapter 5, pages 318–376. Addison-Wesley, São Paulo, Brasil, 2010.

[66] 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.

[67] Yeung, R. W. *Information Theory and Network Coding*, chapter 1, pages 1–4. Springer, Englewood Cliffs, New Jersey, USA, 2008.

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

[69] 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.

Índice Remissivo