

MASTER THESIS

UNIVERSITY OF BERGEN

DEPARTMENT OF INFORMATICS

---

A Comparative Study on Distributed  
Storage and Erasure Coding  
Techniques Using Apache Hadoop  
Over NorNet Core

---

*Author:*  
Maximiliano VELA

*Supervisor:*  
Eirik ROSNES

November 18, 2017



## Acknowledgements

Work exposed in this thesis would not have been possible without the constant assistance, guidance, motivation, and support from my supervisor Eirik Rosnes, currently Senior Research Scientist at Simula@UiB, who has always been available for me whenever I needed him.

I also want to thank Thomas Dreibholz (Simula), Ahmed Elmokashfi (Simula), and Per Simonsen (MemoScale) for their technical support in regards to NorNet Core usage and erasure coding policy handling.

Last but not least, I would not be able to go on without thanking Mari Garaas Løchen and the entire UiB administration for granting me the opportunity of deepening my education in such an upstanding institution.



# Abstract

Both private and public sector organizations are constantly looking for new ways to keep their information safe and accessible at all times. Over the past few decades, replication has always been a reliable way to make sure data is constantly available, even though it has been proven to induce higher costs due to the additional required storage.

Since the early 2000s, erasure codes have been developed as a means to drastically reduce the overhead, while enormously increasing efficiency and providing significant error-correcting capabilities. One of the most well-known erasure coding policies is Reed-Solomon (RS), a highly consistent, reliable, and efficient technique to store and recover data, currently used at Facebook's data centers. Other frequently mentioned policies are Pyramid codes, a variant of Locally Repairable Codes (LRCs) that make use of a pyramid-based scheme to generate additional parity groups for each level, and has been used at Microsoft's Windows Live servers.

Apache Hadoop is an open-source distributed framework used for scalable processing that has recently introduced erasure coding policies to their storage capabilities. NorNet Core (or NorNet Core Testbed<sup>1</sup>), a distributed academic network, will be used as the main scenario to measure, compare, and analyze these different erasure coding policies and their efficiency.

Based on simulations of physically distributed storage, this thesis will show how minimal alterations in commonly known codes (such as RS codes) can converge in a Pyramid-based code that could severely enhance fault-tolerance and performance. Additionally, in a side-to-side comparison, it will be detailed how bigger codes (of higher dimension and length), more often than not, provide a more beneficial trade-off.

---

<sup>1</sup>NorNet Core Testbed website: [www.mntb.no](http://www.mntb.no).



# List of Figures

1.1	CAGR projection from 2016 to 2021. . . . .	1
1.2	Historical global traffic. . . . .	2
2.1	System model. . . . .	6
2.2	Simple erasure coding example. . . . .	7
2.3	General reconstruction scheme for an $n = 6, k = 4$ code. . . . .	8
2.4	Standard 3-way replication. . . . .	9
2.5	XOR(2, 1) example. . . . .	9
2.6	RS encoding phase - $n = 6, k = 4$ . . . . .	10
2.7	MBR-MSR trade-off curve, $n = 15, k = 10, B = 1$ , and $d = n - 1 = 14$ . . . . .	12
2.8	LRC(16, 12, 6) encoding. . . . .	13
2.9	Hierarchical codes. . . . .	13
3.1	Hadoop engine and its modules [1]. . . . .	16
3.2	Hadoop configuration files. . . . .	18
3.3	Hadoop system variables. . . . .	18
3.4	Hadoop environment variables. . . . .	18
3.5	File core-site.xml settings. . . . .	18
3.6	File mapred-site.xml settings. . . . .	19
3.7	File hdfs-core.xml settings. . . . .	19
3.8	Hosts file on Master. . . . .	20
3.9	Adding a new policy to Hadoop, step 1. . . . .	20
3.10	Adding a new policy to Hadoop, step 2. . . . .	21
3.11	Adding a new policy to Hadoop, step 3. . . . .	21
3.12	Apache Hadoop release directory. . . . .	22
3.13	Apache Hadoop compiling executable. . . . .	22
3.14	Apache Hadoop compiling process. . . . .	23
3.15	Apache Hadoop re-compiled file. . . . .	23
4.1	NorNet Core current site list. . . . .	24
4.2	NorNet Core - Norwegian site locations. . . . .	25
4.3	Successful gatekeeper access. . . . .	26
4.4	Successful node access. . . . .	26
4.5	Operating systems and access order. . . . .	27
5.1	Pyramid code construction. . . . .	29
5.2	IPTraF-NG user interface. . . . .	29
5.3	Traffic log sample on a real cluster simulation. . . . .	30
5.4	Node DNS names file. . . . .	30
5.5	Node names file. . . . .	30

6.1	Traffic logs during encoding (in bytes) - RS, $n = 5, k = 3$ .	32
6.2	Participating nodes - RS, $n = 5, k = 3$ .	33
6.3	Cluster status before reconstruction - RS, $n = 5, k = 3$ .	33
6.4	Traffic logs during reconstruction (in bytes) - RS, $n = 5, k = 3$ .	34
6.5	Cluster status after reconstruction - RS, $n = 5, k = 3$ .	34
6.6	Traffic logs during encoding (in bytes) - RS, $n = 11, k = 8$ .	34
6.7	Participating nodes - RS, $n = 11, k = 8$ .	35
6.8	Cluster status before reconstruction - RS, $n = 11, k = 8$ .	35
6.9	Traffic logs during reconstruction (in bytes) - RS, $n = 11, k = 8$ .	36
6.10	Cluster status after reconstruction - RS, $n = 11, k = 8$ .	36
6.11	Participating nodes - RS, $n = 14, k = 10$ .	37
6.12	Traffic logs during encoding (in bytes) - RS, $n = 14, k = 10$ .	37
6.13	Cluster status before reconstruction - RS, $n = 14, k = 10$ .	38
6.14	Traffic logs during reconstruction (in bytes) - RS, $n = 14, k = 10$ .	38
6.15	Cluster status after reconstruction - RS, $n = 14, k = 10$ .	39
6.16	Traffic logs during encoding (in bytes) - Pyramid, $n = 6, k = 3$ .	40
6.17	Participating nodes - Pyramid, $n = 6, k = 3$ .	40
6.18	Cluster status before reconstruction - Pyramid, $n = 6, k = 3$ .	41
6.19	Traffic logs during reconstruction (in bytes) - Pyramid, $n = 6, k = 3$ .	41
6.20	Cluster status after reconstruction - Pyramid, $n = 6, k = 3$ .	43
6.21	Traffic logs during encoding (in bytes) - Pyramid, $n = 12, k = 8$ .	43
6.22	Participating nodes - Pyramid, $n = 12, k = 8$ .	44
6.23	Cluster status before reconstruction - Pyramid, $n = 12, k = 8$ .	44
6.24	Traffic logs during reconstruction (in bytes) - Pyramid, $n = 12, k = 8$ .	45
6.25	Cluster status after reconstruction - Pyramid, $n = 12, k = 8$ .	46
6.26	Participating nodes - Pyramid, $n = 15, k = 10$ .	47
6.27	Traffic logs during encoding (in bytes) - Pyramid, $n = 15, k = 10$ .	47
6.28	Cluster status before reconstruction - Pyramid, $n = 15, k = 10$ .	48
6.29	Traffic logs during reconstruction (in bytes) - Pyramid, $n = 15, k = 10$ .	49
6.30	Cluster status after reconstruction - Pyramid, $n = 15, k = 10$ .	50
6.31	Network traffic comparison.	52
6.32	Duration comparison.	53





# List of Tables

6.1	Results summary - Network traffic and duration. . . . .	51
6.2	Network traffic comparison during encoding. . . . .	51
6.3	Network traffic comparison during reconstruction. . . . .	51
6.4	Duration comparison during encoding. . . . .	52
6.5	Duration comparison during reconstruction. . . . .	52
6.6	Repair comparison. . . . .	53



# Contents

Acknowledgements . . . . .	i
Abstract . . . . .	iii
List of Figures . . . . .	v
List of Tables . . . . .	viii
Contents . . . . .	x
<b>1 Introduction</b>	<b>1</b>
1.1 Objective . . . . .	2
1.2 Thesis Organization . . . . .	3
<b>2 Distributed Storage</b>	<b>4</b>
2.1 Linear Codes . . . . .	4
2.1.1 Maximum Distance Separable (MDS) Codes . . . . .	6
2.2 System Model . . . . .	6
2.3 Erasure Codes . . . . .	7
2.4 Repair Process . . . . .	7
2.5 Traditional Codes Used in Distributed Storage . . . . .	8
2.5.1 Replication . . . . .	8
2.5.2 XOR . . . . .	9
2.5.3 RS Codes . . . . .	9
2.6 Regenerating Codes . . . . .	11
2.7 LRCs . . . . .	12
2.7.1 Pyramid and Hierarchical Codes . . . . .	13
2.8 Tested and Used Schemes . . . . .	14
<b>3 Hadoop</b>	<b>16</b>
3.1 Overview . . . . .	16
3.2 Requirements . . . . .	17
3.3 Configuration . . . . .	17
3.4 Adding New Erasure Coding Policies to Hadoop . . . . .	20
3.5 Recompiling Hadoop's Source Code . . . . .	21
<b>4 NorNet Core</b>	<b>24</b>
4.1 Description . . . . .	24
4.2 Access . . . . .	25
<b>5 Storage Simulations and Measurements</b>	<b>28</b>
5.1 Overall Concepts and Methodology Used . . . . .	28
5.2 Software Used . . . . .	29
5.3 Traffic Log-Harvesting Script . . . . .	30

5.4	Participating Nodes . . . . .	31
<b>6</b>	<b>Results</b>	<b>32</b>
6.1	RS Codes . . . . .	32
6.1.1	RS(5,3) . . . . .	32
6.1.2	RS(11,8) . . . . .	34
6.1.3	RS(14,10) . . . . .	36
6.2	Pyramid Codes . . . . .	39
6.2.1	PYR(6,3) . . . . .	40
6.2.2	PYR(12,8) . . . . .	43
6.2.3	PYR(15,10) . . . . .	46
6.3	Results Summary . . . . .	50
<b>7</b>	<b>Conclusion and Further Work</b>	<b>55</b>
<b>8</b>	<b>Appendix - Code Snippets</b>	<b>57</b>
8.1	Log-Harvesting Script . . . . .	57
8.2	Hadoop's Default Cauchy Matrix Generator . . . . .	58
8.3	Hadoop's Modified Cauchy Matrix Generator . . . . .	59
8.4	Hadoop's Modified Pyramid Policy . . . . .	59
	<b>References</b>	<b>62</b>



# Chapter 1

## Introduction

Today's applications and services are using more and more data every year, naturally having social networks in the lead. Livestreaming, media sharing, personal data, and interactions sent from and between users demand immense amounts of disk space meant to store data either temporarily or permanently. A report regarding Facebook's storage upgrade in April 2014 [2] points out that they were at the time receiving a daily data income rate of 600 TB from more than 1.9 billion users.

Network traffic is also increasing at an enormous rate every year. One of the latest white papers published by Cisco [3] describes an astonishing series of Internet traffic facts:

- Yearly IP traffic in 2016 has come up to 1.2 Zettabytes ( $10^{21}$  bytes, or 1000 Exabytes), and will continue to reach 3.3 ZB per year by 2021.
- Between 2016 and 2021, mobile data traffic will increase sevenfold.
- From 2016 to 2021, IP traffic will grow at a Compound Annual Growth Rate (CAGR) of 24 percent, as depicted in Figure 1.1.

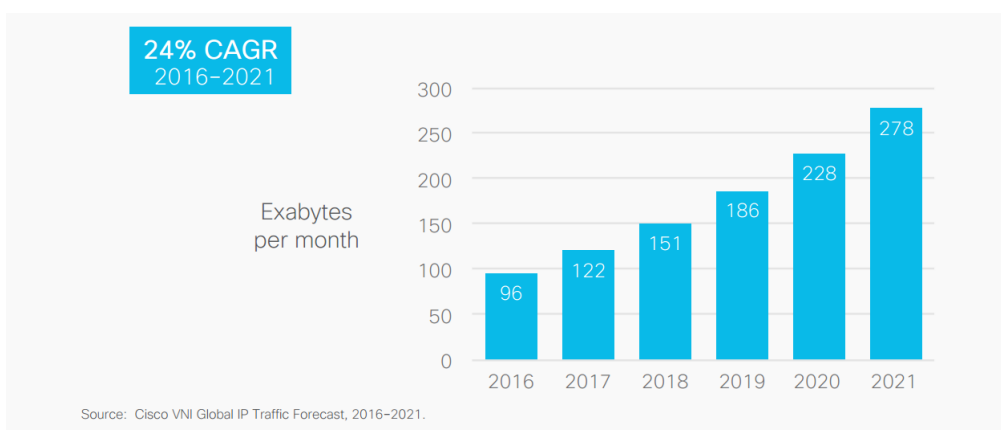


Figure 1.1: CAGR projection from 2016 to 2021.

- Monthly IP traffic per capita accounted for 13 GB in 2016, and will continue to grow up to 35 GB by 2021.
- Global Internet traffic reached 26 600 GB per second, and will amount to 105 800 GB per second in 2021, as shown in Figure 1.2.

Year	Global Internet Traffic
1992	100 GB per day
1997	100 GB per hour
2002	100 GB per second
2007	2,000 GB per second
2016	26,600 GB per second
2021	105,800 GB per second

Source: Cisco VNI, 2017.

Figure 1.2: Historical global traffic.

The first concern for both large and smaller companies is to be able to cover up the storage requirements as they continue to grow over time, but it is also very important to use the available storage in an effective manner. Data replication has been widely used over the years but more specifically during the 80s and 90s, when overly inflated budgets and mostly unused, redundant information were the only way to keep the data available at all times.

Distributed storage has been introduced to the industry during the mid 90s, and it continues to be on the rise as more and more companies choose this concept not only due to its indisputable flexibility, but also its ability to significantly reduce costs and increase processing speed [4]. A cluster of nodes is often treated as a single storage unit (using standard infrastructure, drivers and network), while combining RAM and CPU capacities from several servers into a common pool. As cluster size grows, so does the processing speed - additionally, servers are required to store less data, and the transfer rate per node decreases drastically. A master node is often necessary to command these individual efforts and increase fault-tolerance within the cluster.

In a typical distributed storage system, participating servers can either be located in the same data center or within geographically distant locations without affecting their behavior. Some businesses rely on cloud storage companies that take care of such details and provide the infrastructure they require: some examples are Amazon AWS, Windows Azure, Rackspace, and IBM SmartCloud.

On top of this approach, erasure codes have been developed as an attempt to reduce storage overhead and increase fault-tolerance thanks to their recovery capabilities. These codes allow distributed systems to entirely re-generate missing parts of the file structure (i.e., parts of a file, a database, or even entire disks) without requiring any contact with the detached node. Depending on the properties of the erasure code used, it could also retrieve information stored in several different nodes that can no longer be reached with relative ease.

## 1.1 Objective

The initial objective in this academic work is to determine whether Apache Hadoop usage is viable over NorNet Core's platform. If so, the main objective is then to analyze and compare erasure coding policies (preferably more than just the ones officially provided by Apache), assess their efficiency, and evaluate if previous research



correlates to our pragmatic results.

## 1.2 Thesis Organization

- **Chapter 2:** Overview of linear and erasure codes, a global system model definition along with a brief explanation of some of the most relevant erasure coding techniques.
- **Chapter 3:** General Hadoop concepts, a review of some of its modules and more technical details regarding requirements and configuration used in this thesis.
- **Chapter 4:** NorNet Core description, list of participating institutions and their nodes, as well as a short guide on how to access NorNet Core.
- **Chapter 5:** Some guidelines regarding the way we obtained results shown later in the thesis, software used to gather measurements, followed by the list of NorNet Core nodes that have been used in our experiments.
- **Chapter 6:** Detailed information about the results we obtained, cluster statuses throughout simulations, and measurements harvested.
- **Chapter 7:** Conclusion regarding results obtained, and suggested future work.
- **Chapter 8:** Appendix of code fragments used in this thesis.

# Chapter 2

## Distributed Storage

Distributed storage and computing is a wide concept that involves performing tasks in a parallel manner, combining the processing power and storage capabilities of multiple servers. This scheme is being increasingly used in the industry, as it allows companies to make a more efficient use of the hardware they own at a lower cost, yielding almost no disadvantages.

Oftentimes, when distributed storage systems are combined with erasure coding techniques, they are designed based on a trade-off within a list of properties, where not all of them can be fully achieved simultaneously. The most important properties are:

- Resiliency to disk failures or fault-tolerance.
- Storage overhead or efficiency.
- Download complexity.
- Repair cost of a node or repair bandwidth.
- Upgrade cost.
- Security level (e.g., eavesdropping and data-tampering).

The most common example is erasure coding schemes that have high resiliency at the expense of a higher repair cost (typically, Reed-Solomon (RS) codes), whereas other techniques improve upon the required repair bandwidth (number of symbols needed to be downloaded to repair a single node) while sometimes conditioning the fault-tolerance of the code in specific scenarios.

Some of these properties will be discussed in this chapter, along with a general definition of both linear and erasure codes.

### 2.1 Linear Codes

A linear code  $\mathcal{C}$  can be defined as a  $k$ -dimensional subspace of the vector space  $\mathbb{F}_q^n$ , where  $n$  is the length of the code,  $k$  is the dimension, and  $\mathbb{F}_q$  is the finite field over which the code is defined. The codes are defined as  $q$ -ary codes, where  $q$  is a prime number or a prime power. The linear space  $\mathcal{C}$  can be represented as a set of codewords, often found in row combinations of a generator matrix  $G$ . This matrix is

said to be in standard form (and the code systematic) whenever its leftmost columns correspond to an identity matrix  $I_k$  of size  $k \times k$ , following the definition:

$$G = [I_k | P] = \left( \begin{array}{cccc|ccc} g_{1,1} & g_{1,1} & \cdots & g_{1,k} & g_{1,k+1} & \cdots & g_{1,n} \\ g_{2,1} & g_{2,2} & \cdots & g_{2,k} & g_{2,k+1} & \cdots & g_{2,n} \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ g_{k,1} & g_{k,2} & \cdots & g_{k,k} & g_{k,k+1} & \cdots & g_{k,n} \end{array} \right).$$

From this generator matrix we can obtain a parity check matrix  $H$ , which contains the coefficients of the parity check equations. These coefficients show how certain linear combinations of the coordinates of each codeword  $c \in \mathcal{C}$  equal zero, as defined below:

$$H = [-P^T | I_{n-k}] = \left( \begin{array}{cccc|ccc} h_{1,1} & h_{1,2} & \cdots & h_{1,k} & h_{1,k+1} & h_{1,k+2} & \cdots & h_{1,n} \\ h_{2,1} & h_{2,2} & \cdots & h_{2,k} & h_{2,k+1} & h_{2,k+2} & \cdots & h_{2,n} \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ h_{n-k,1} & h_{n-k,2} & \cdots & h_{n-k,k} & h_{n-k,k+1} & h_{n-k,k+2} & \cdots & h_{n-k,n} \end{array} \right),$$

where  $(\cdot)^T$  denotes the transpose of its argument. As a result we can converge in a set of parity check equations:

$$\begin{aligned} h_{1,1}c_1 + h_{1,2}c_2 + \cdots + h_{1,n-1}c_{n-1} + h_{1,n}c_n &= 0 \\ h_{2,1}c_1 + h_{2,2}c_2 + \cdots + h_{2,n-1}c_{n-1} + h_{2,n}c_n &= 0 \\ \cdots + \cdots + \cdots + \cdots &= 0 \\ h_{n-k,1}c_1 + h_{n-k,2}c_2 + \cdots + h_{n-k,n-1}c_{n-1} + h_{n-k,n}c_n &= 0, \end{aligned}$$

where  $c = (c_1, c_2, \dots, c_n)$  denotes a codeword from  $\mathcal{C}$ .

As an example, we can take a binary linear code of dimension  $k = 3$  and length  $n = 5$ . A generator matrix  $G$  could have the following structure:

$$G = \begin{pmatrix} 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 \end{pmatrix},$$

with parity check matrix

$$H = \begin{pmatrix} 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 \end{pmatrix}.$$

Identifying each coefficient of a codeword as  $c_i$  (the codeword  $c = (c_1, c_2, \dots, c_5)$ ) we obtain the check equations:

$$\begin{aligned} c_1 + c_2 + c_4 &= 0 \\ c_2 + c_3 + c_5 &= 0. \end{aligned}$$

Associating this simple equation set to the field of regenerating codes, we can claim that in the case of a missing coefficient  $c_2$ , we can re-obtain it by replacing it with the binary sum of  $c_1$  and  $c_4$  (or the binary sum of  $c_3$  and  $c_5$ ). As  $n$  gets bigger, so does the equation set size, thus allowing the recovery of several missing coordinates simultaneously.

### 2.1.1 Maximum Distance Separable (MDS) Codes

MDS codes or maximum distance separable codes are a type of  $(n, k)$  linear codes of minimum Hamming distance  $d_{\min}$  that meets the Singleton bound  $d_{\min} \leq n - k + 1$ , where the minimum Hamming distance is the minimum number of positions in which two distinct codewords differ. For linear codes  $d_{\min}$  is equal to the minimum Hamming weight over all non-zero codewords, where the Hamming weight of a codeword is the number of non-zero entries it contains. As the error correcting capabilities of linear codes are related to the minimum Hamming distance, MDS codes attain the highest correction capacity. An  $(n, k)$  linear code  $\mathcal{C}$  fulfills the MDS condition if, and only if, one of the following statements are true:

1. The minimum Hamming distance of  $\mathcal{C}$  is  $n - k + 1$ .
2. The rank of the parity check matrix  $H$  is  $n - k$ , and every  $(n - k) \times (n - k)$  sub-matrix is full-rank.
3. The rank of the generator matrix  $G$  is  $k$ , and every  $k \times k$  sub-matrix is full-rank.

## 2.2 System Model

When referring to linear and erasure codes there is a necessity to describe a general system model where all these schemes take place. We can take for instance a distributed storage system that stores a set of  $f$  files  $D^1, \dots, D^f$ , where each of these files  $D^m = [d_{i,j}^m]$ ,  $m = 1, \dots, f$ , is a  $\delta \times k$  matrix over  $\mathbb{F}_q$ , where  $\delta$  and  $k$  are positive integers and  $q$  some prime number or prime power. Files are split into a list of  $\delta$  stripes and encoded using a linear code. Let  $d_i^m = (d_{i,1}^m, d_{i,2}^m, \dots, d_{i,k}^m)$ ,  $i = 1, \dots, \delta$ , be a message vector that is encoded by an  $(n, k)$  linear code  $\mathcal{C}$  over  $\mathbb{F}_q$ , into a length- $n$  codeword  $c_i^m = (c_{i,1}^m, c_{i,2}^m, \dots, c_{i,n}^m)$ , where  $c_{i,j}^m \in \mathbb{F}_q$ . The  $\delta f$  generated codewords  $c_i^m$  are allocated in the array  $C = ((c_1^1)^T | \dots | (c_\delta^1)^T | \dots | (c_\delta^f)^T)^T$  of dimension  $\delta f \times n$ , where  $(v_1 | \dots | v_{\delta f})$  denotes the concatenation of column vectors  $v_1, \dots, v_{\delta f}$ . The symbols  $c_{i,j}^m$  for a fixed  $j$  are stored on the  $j$ -th node. If the code  $\mathcal{C}$  is systematic, we assume that the first  $k$  code symbols are message symbols and the subsequent  $n - k$  symbols are parity symbols. This description is depicted on Figure 2.1.

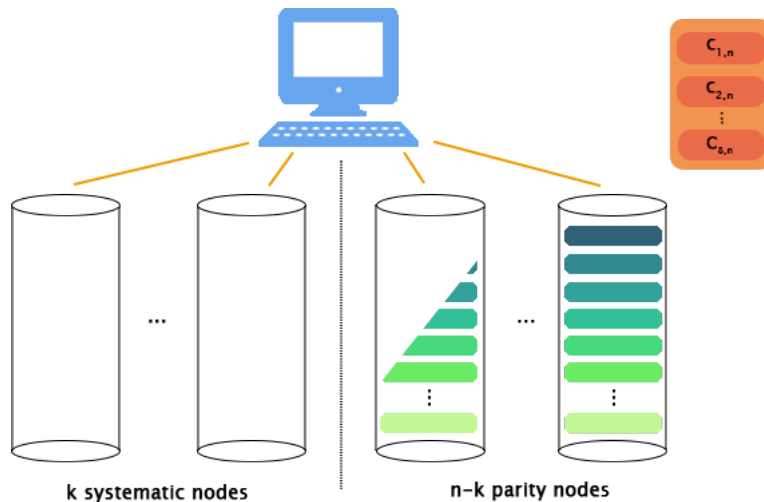


Figure 2.1: System model.

## 2.3 Erasure Codes

Within the field of information theory, an erasure code is a technique based on linear codes that splits data into fragments that are used to generate additional parity pieces. This expanded message can be later used to recover missing fragments from any combination of remaining pieces depending on the code structure. Erasure codes are often mentioned as a type of Forward Error Correction (FEC), used to detect and control errors in data transmission over noisy or unreliable channels.

More specifically, an erasure code is able to encode  $k$  data parts of a certain size into  $n$  chunks after adding up parity pieces (see Figure 2.2), and each set of  $n$  symbols is often referred to as a stripe. The goal of this code, of parameters  $(n, k)$  is to regenerate up to  $n - k$  missing or corrupted chunks from any remaining  $k$  parts.

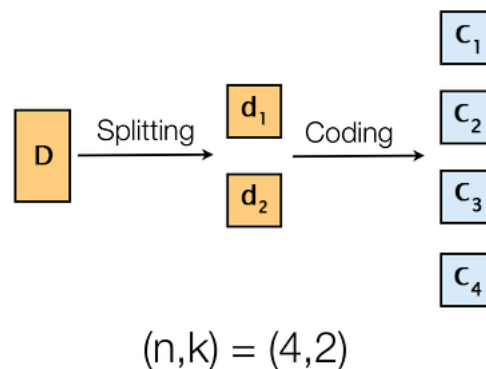


Figure 2.2: Simple erasure coding example.

In the example above, taking for instance an original file of size 1 MB, would result in a total size of  $1 \text{ MB} \times \frac{4}{2} = 2 \text{ MB}$  when encoded, or a 100% overhead.

Nowadays, there are several erasure coding techniques, each with its very own parameters, functionality, logic, and complexity. The following sections provide a quick introduction to a few erasure coding schemes and their uses, as well as the general repair process.

## 2.4 Repair Process

After a file has gone through the encoding phase and subsequent parity blocks have been generated, each segment is typically stored within distantly located nodes that maintain constant communication between each other. On a real scenario (for example during Hadoop's encoding), each block is stored on a random cluster location, meaning that each node is independent of the block positions it contains.

When a node goes down, either temporarily or permanently, and there exists another available node, for RS codes and regenerating codes (see Section 2.6 below) a minimum of  $k$  blocks from each stripe is transferred over to the newly joining node, which is capable of re-generating the originally missing block. Other erasure codes, however, like Locally Repairable Codes (LRCs) (see Section 2.7 below), may require less than  $k$  blocks. This reconstruction process can be depicted directly below in Figure 2.3 for an example code.

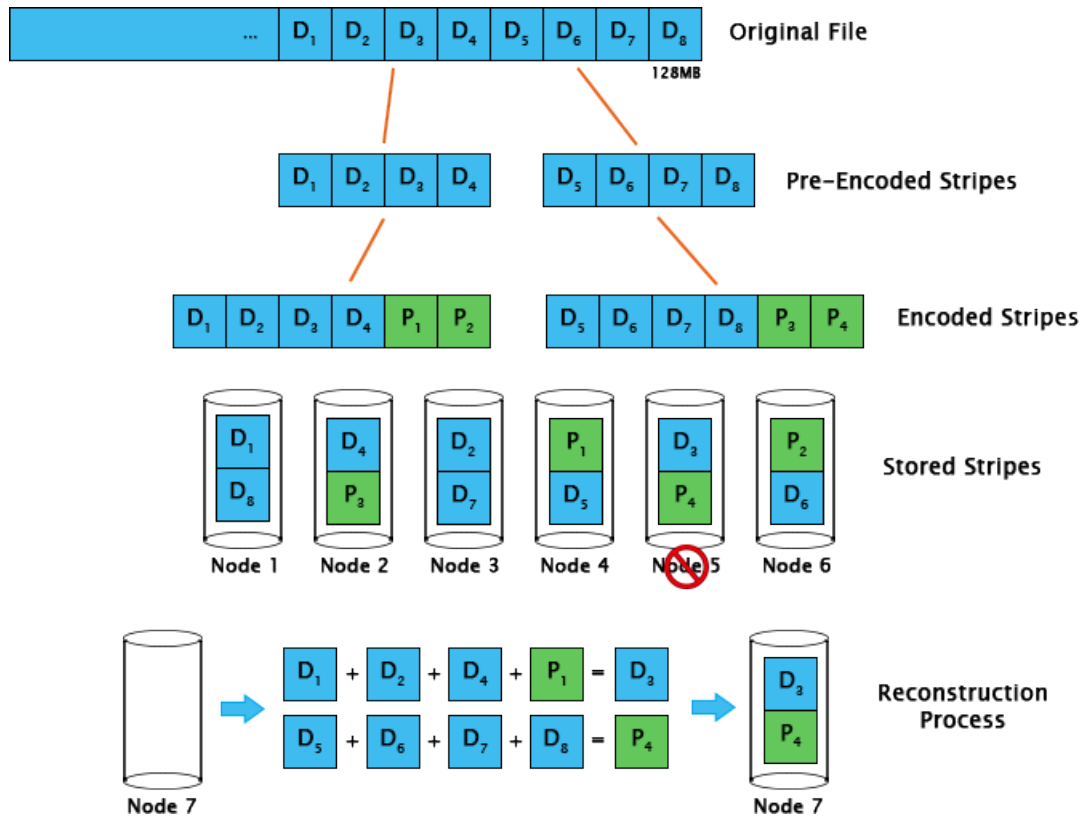


Figure 2.3: General reconstruction scheme for an  $n = 6, k = 4$  code.

## 2.5 Traditional Codes Used in Distributed Storage

The following subsections give an overview of traditional codes used in distributed storage systems.

### 2.5.1 Replication

Standard replication consists of generating additional data copies and storing each of these in different physical or logical locations. This procedure lacks of an encoding phase as no additional parity blocks are generated, and yields an overhead of 100% per replication factor. Even though it provides a clear negative tradeoff in terms of storage efficiency, up to 4-way replication has been widely used in the industry in the last decade. Formal comparison between this scheme and erasure coding techniques has been analyzed in [5,6]. A simple 3-way replication example can be found below in Figure 2.4.

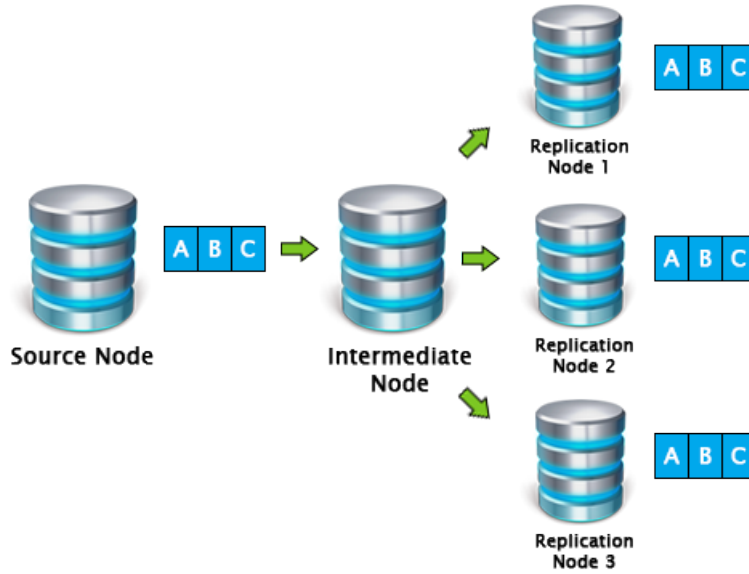


Figure 2.4: Standard 3-way replication.

## 2.5.2 XOR

XOR-based encoding is arguably one of the most basic erasure coding techniques available. Recent research work [7, 8] state that these schemes can be computationally more efficient than MDS codes, but still offer a negative trade-off in terms of performance, space-efficiency, and fault-tolerance.

Having  $k$  symbols of size  $s$  bits, an additional parity symbol is generated by XOR-ing the previous data symbols, thus reaching  $n = k + 1$  total symbols. This encoding is often referred to as XOR( $n, k$ ), and the most common implementation is XOR(2, 1), depicted in Figure 2.5.

$$\begin{aligned}
 \mathbf{B}_1 &= [1\ 1\ 0\ 0\ 1\ 0\ 1\ 1] \\
 \mathbf{B}_2 &= [1\ 0\ 1\ 0\ 0\ 1\ 0\ 0] \\
 \mathbf{B}_p = \mathbf{B}_1 \oplus \mathbf{B}_2 &= [0\ 1\ 1\ 0\ 1\ 1\ 1\ 1]
 \end{aligned}$$

Figure 2.5: XOR(2, 1) example.

Even though XOR codes will always provide a better storage overhead than standard replication, its simplicity will inevitably make it much less trustworthy than more robust codes such as RS codes or LRCs. Different authors have proposed modified XOR-based policies, such as Hitchhiker-XOR/XOR+ [9], claiming to significantly enhance performance.

## 2.5.3 RS Codes

RS codes were introduced in 1960 by Irving S. Reed and Gustave Solomon and they have very wide applications, ranging from CD/DVD storage to satellite transmis-

sions. They are a special class of MDS codes, and are often specified as an  $RS(n, k)$  code with  $s$ -bits symbols.

In the encoding phase, data is split into sets of  $k$  data symbols of size  $s$ , and parity symbols are added to make a codeword of size  $n$ . As a result,  $n - k$  parity symbols of size  $s$  will be generated. When these codewords are decoded, up to  $n - k$  erased symbols can be corrected.

Typically, the first section of the generator matrix  $G$  corresponds to an identity matrix of size  $k \times k$ . Contiguously there must be a smaller structure of size  $k \times (n - k)$  with coefficients ( $g_{15}, g_{25}, \dots, g_{45}, g_{16}, g_{26}, \dots, g_{46}$  on Figure 2.6) that will allow the codeword encoding later on. For an RS code, the resulting matrix of size  $k \times (n - k)$  can be a *Cauchy* matrix. On the decoding phase, the inverse process is performed in order to retrieve the original raw data.

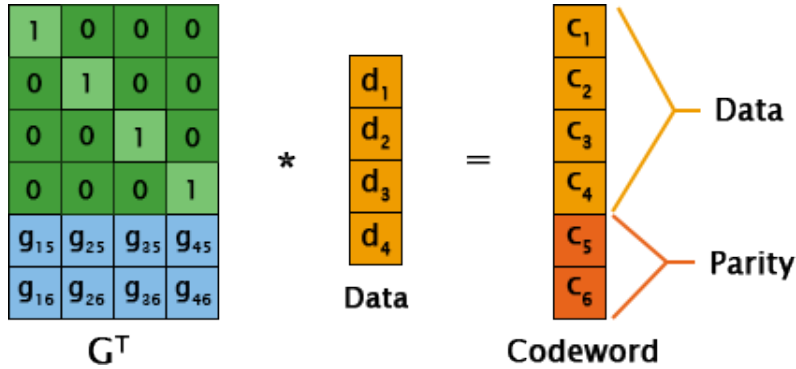


Figure 2.6: RS encoding phase -  $n = 6, k = 4$ .

The storage overhead for RS codes (as well as for general linear codes) is  $\frac{n}{k}$ . For instance, given a file of size 2 GB and an RS code with parameters  $(5, 3)$ :

$$\text{Overhead} = \frac{5}{3} = 1.66.$$

$$\text{File size} = \frac{5}{3} \times 2 \text{ GB} = 3.33 \text{ GB}.$$

The most common RS code implementations for distributed storage, including some that will be studied in this academic work are  $RS(5, 3)$ ,  $RS(9, 6)$ , and  $RS(14, 10)$ . These three configurations can be found on Hadoop's default erasure coding policies from version 3.0.0 onwards. Some slight parameter variation will also be included for analysis and comparison.

### RS Hadoop Example

A framework based on Hadoop's file system (HDFS), Xorbas, has been previously used to compare RS codes with other policies [10]. Take for instance an RS code with the parameters specified in the figure above. A real example of a Cauchy matrix generated using Hadoop's RS encoding phase can be found below:

$$G = \begin{pmatrix} 1 & 0 & 0 & 0 & 71 & -89 \\ 0 & 1 & 0 & 0 & 122 & -70 \\ 0 & 0 & 1 & 0 & -89 & 71 \\ 0 & 0 & 0 & 1 & -70 & 122 \end{pmatrix}.$$



Although this matrix is often depicted as a two-dimensional array, Hadoop generates and transmits it as a one-dimensional array, thus locating the identity matrix between positions 0 and  $k^2 - 1$ . Subsequent positions will contain the randomly generated coefficients, between positions  $k^2$  and  $nk - 1$ , and will refer to two incrementing indexes  $i$  and  $j$ , precisely in the form of  $\frac{1}{i+j}$ . Source code from Hadoop's default Cauchy matrix generation procedure can be found under Section 8.2.

## 2.6 Regenerating Codes

During storage failures, codes must be able to regenerate corrupt or missing nodes and resume normal activities. A traditional MDS code (like an RS code) must typically connect to  $k$  nodes, download the entirety of their contents and re-encode the data in order to extract the corrupted blocks. This scheme may result in a lack of efficiency, as the entire stripe must be downloaded just to store a small fraction of the structure within the new node. Regenerating codes, introduced by Dimakis *et al.* [11, 12] are an attempt to avoid this issue and significantly reduce the repair bandwidth, defined here as the number of symbols needed to be downloaded to repair a single node. We remark that there has been some recent work on the efficient repair of RS codes, showing that RS codes can indeed be more efficiently repaired than what was first believed. See, for instance, [13] and references therein.

Given an amount  $\alpha$  of information (or symbols) stored on each node, and an amount  $\beta$  of information (or symbols) sent from  $k \leq d \leq n - 1$  nodes during the recovery process, we can identify the total repair bandwidth as  $\gamma = d\beta$ . According to [11], it is proven that the system capacity  $C_{k,d}(\alpha, \gamma)$  (or the maximum file size in terms of symbols that can be stored on the system) is

$$C_{k,d}(\alpha, \gamma) = \sum_{j=0}^{k-1} \min \left( \alpha, \frac{d-j}{d} \gamma \right).$$

The expression above gives an inherent trade-off between storage capacity per node  $\alpha$  and repair bandwidth  $\gamma$ . Setting  $C_{k,d}(\alpha, \gamma) = B$ , explicit expressions for the two boundary points of this trade-off curve can be obtained. The minimum storage regenerating (MSR) point is obtained through minimizing  $\alpha$  and then  $\gamma$  to obtain:

$$\begin{cases} \alpha = \frac{B}{k} \\ \gamma = \frac{dB}{k(d-k+1)} \end{cases}.$$

Similarly, we can reach the minimum bandwidth regenerating (MBR) point by minimizing in the opposite order, and get

$$\begin{cases} \alpha = \frac{2dB}{k(2d-k+1)} \\ \gamma = \frac{2dB}{k(2d-k+1)} \end{cases}.$$

These two points are located at the two ends of the trade-off curve [11], depicted in Figure 2.7 for  $n = 15$ ,  $k = 10$ ,  $B = 1$ , and  $d = n - 1 = 14$ .

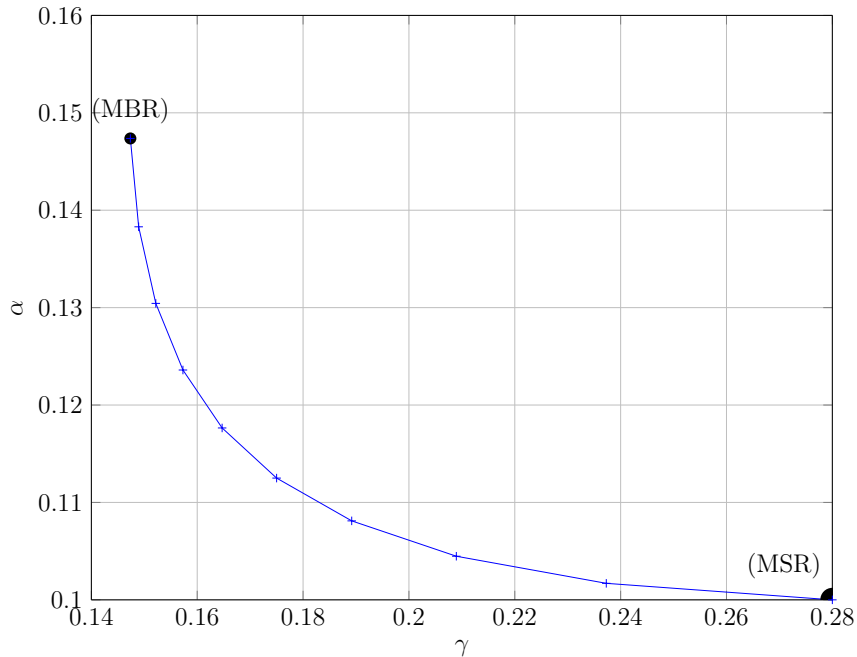


Figure 2.7: MBR-MSR trade-off curve,  $n = 15$ ,  $k = 10$ ,  $B = 1$ , and  $d = n - 1 = 14$ .

Note that the trade-off curve described above is for *functional* repair, where there is no requirement that the reconstructed node shall be an identical copy of the failed node, but nevertheless provide equal overall fault-tolerance. Under *exact* repair, however, such a requirement is imposed and codes operating on the curve are known to not exist at essentially all interior points [14]. See also [15] for code constructions operating between the MBR and the MSR points with the exact repair property. At both the MBR [16] and the MSR [16, 17] points, however, code constructions under exact repair for all values of  $(n, k, d)$  are known to exist. Related work finding minimum required storage in this type of codes can be found in [18, 19].

## 2.7 LRCs

As described in one of the papers that first introduced this type of codes [20]: An  $(n, r, d_{\min}, M, \alpha)$ -LRC is a code that takes a file of size  $M$  bits, encodes it into  $n$  coded symbols of size  $\alpha$  bits such that any of these  $n$  coded symbols can be reconstructed by accessing and processing at most  $r$  other code symbols. Moreover, the minimum distance of the code is  $d_{\min}$ , i.e., the file of size  $M$  bits can be reconstructed by accessing any  $n - d_{\min} + 1$  of the  $n$  coded symbols. LRC implementations split data into several subgroups of a certain size, generate local parity fragments for each of them and global parity blocks for the stripe as a whole. LRCs are important for applications where not only the repair bandwidth, but also the number of nodes needed to be contacted during repair matters. The number of nodes needed to be contacted during repair is often referred to as the *repair locality/access*. In the following, to simplify notation, an  $(n, k, r)$ -LRC, sometimes denoted as  $\text{LRC}(n, k, r)$ , is code of length  $n$  that takes as input  $k$  information symbols, such that any of its  $n$  output coded symbols can be recovered by accessing and processing at most  $r$  other code symbols. The minimum distance  $d_{\min}$  of an  $(n, k, r)$ -LRC is upper-bounded [21]

by

$$d_{\min} \leq n - k - \left\lceil \frac{k}{r} \right\rceil + 2$$

which can be seen as a modification of the Singleton bound. From the formula above, a trade-off between locality and fault-tolerance can be observed. Any LRC achieving the upper bound above is called an *optimal* LRC. Designing optimal LRCs for any triple  $(n, k, r)$  that are easy to implement is an active research area. Further analysis regarding LRCs can be found in [10, 22, 23].

Figure 2.8 shows a typical LRC(16, 12, 6) scenario, used in Windows Azure Storage [24, 25]. Fragments  $(c_1, c_2, \dots, c_6)$  and  $(c_7, c_8, \dots, c_{12})$  correspond to data fragments, having  $LP_1$  and  $LP_2$  as local parities for each subgroup. Additionally,  $GP_1$  and  $GP_2$  are derived global parities in similar fashion as in the RS encoding phase.

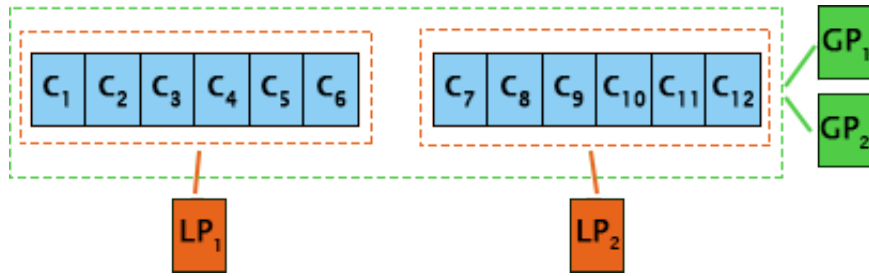


Figure 2.8: LRC(16, 12, 6) encoding.

### 2.7.1 Pyramid and Hierarchical Codes

Pyramid and hierarchical codes are special classes of LRCs. We illustrate the concept by an example. Given a set of symbols  $(c_1, c_2, \dots, c_8)$ , a hierarchical code would first start by generating sets of local parities of size  $s = 2$ . We can divide our list of symbols into symbol subgroups  $c_1, c_2; c_3, c_4; \dots; c_7, c_8$  and encode each fragment pair, obtaining local redundancy. Subsequently, a new parity group of size  $s = 4$  must be generated as a means to allow redundancy to a higher level. The set of symbols will now be split into two subgroups  $c_1, c_2, c_3, c_4; c_5, c_6, c_7, c_8$  and later encoded. This method will go on creating local redundancies on a bigger scale every time, as long as the code size allows it. This iterative process is depicted below on Figure 2.9.

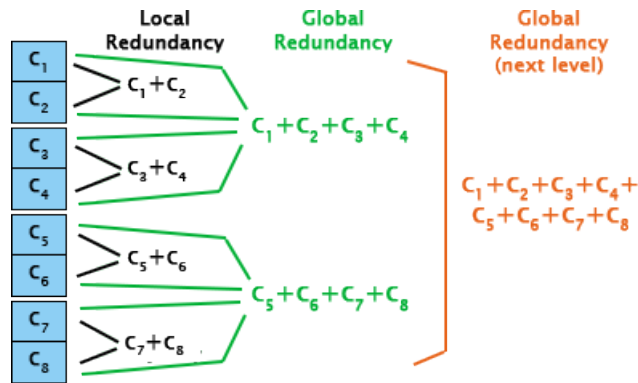


Figure 2.9: Hierarchical codes.

In this hierarchical scheme, we can easily retrieve single erased symbols by using the remains of its local group and its corresponding local redundancy. For example, when computing  $c_1 + c_2$  and  $c_2$  we can easily obtain  $c_1$ . Additionally, if two simultaneous erasures happen where a local redundancy goes missing as well as a single symbol (say  $c_1$  and  $c_1 + c_2$ ), we can replace the remaining symbol on the first-level global redundancy ( $c_1 + c_2 + c_3 + c_4$  in this case) to obtain  $c_1$ , and then recover the local redundancy. However, if a complete local redundancy group goes missing ( $c_1$  and  $c_2$ ), recovery is no longer possible.

As opposed to hierarchical codes, pyramid codes have a top-down scheme in which local redundancies are generated from a single global parity. Take for instance an RS(6, 4) code. We have four data and two parity symbols, namely  $d_1, d_2, d_3, d_4, p_1, p_2$ . From this sequence, these codes would obtain two parity fragments from  $p_1$ , namely  $p_{1a}$  and  $p_{1b}$ . These two newly generated fragments can be used to obtain the original parity symbol, as  $p_{1a} + p_{1b} = p_1$ . Each of these acts as a parity symbol for each half of the data fragment total. In this case,  $p_{1a}$  can be used along with  $d_1$  and  $d_2$  to retrieve missing components, and similarly  $p_{1b}$  can be used with  $d_3$  and  $d_4$ . If the code size allows it, we can generate a new level of fragments related to a smaller data portion (i.e.,  $p_{1aa}, p_{1ab}$ , and so on), which in our case would not be possible as data groups become too small to generate local redundancy.

The aim of these types of codes is to allow reconstructions using the smallest possible portion of symbols, thus reducing the bandwidth to the very minimum, while keeping the regenerating capabilities more robust codes have (e.g., MSR, MBR, and RS codes) through global parities in the worst-case scenarios. More information and examples of these types of codes can be found in [26].

## 2.8 Tested and Used Schemes

Most of the schemes described previously have been or are used to some extent in the industry. Those found on top of the list are also the most tested, analyzed, and proven to yield a more favorable trade-off in terms of download complexity, recovery bandwidth, fault-tolerance and/or storage requirements. These are:

- **RS**

- RS(14, 10), Facebook: in [27], this code (along with the rest included in this paper) is compared with HACFS, an implementation that uses two different product code constructions meant to be used independently whether information retrieved is either hot or cold, depending on how often it is accessed. These two code constructions are namely a *fast* code (PC(2×5)) and a *compact* code (PC(6×5)), where the *fast* code aims to have a lower recovery cost at the expense of a higher storage overhead, and the *compact* code attempts to achieve the exact opposite. According to [27], this code attained the highest degraded read latency and reconstruction time, while achieving a lower storage overhead.
- RS(9, 6), Google Colossus: in [27] it is also shown how this smaller code provided a much better read latency and significantly lower recovery duration, although providing the overall largest storage overhead.

- **LRC**

- LRC(20, 12, 2): this code is referred to in [27] as  $\text{LRC}_{\text{fast}}$ , including 6 local parity blocks as opposed to its most commonly known configuration with only two, mentioned below.
- LRC(16, 12, 6), Azure Storage: previously cited scientific paper by Xia *et al.* [27] describes how this LRC policy provided better storage overhead as the rest of the codes, but otherwise obtained similar results in terms of reconstruction time and degraded read latency. In [25], this code is compared to RS policies of a similar size, showing how fault-tolerance capabilities can be traded off as a means to obtain a lower repair bandwidth.

- **Pyramid**

- PYR(12, 8), Microsoft's Windows Live: paper [28] by Microsoft compares this Pyramid scheme with an MDS policy with the same  $k$  parameter (identified as the MDS code (11, 8)), showing how it can enhance average read overhead and allowing up to 4 simultaneous block reconstructions (as opposed to the MDS code, where up to 3 are possible).

# Chapter 3

## Hadoop

Apache Hadoop is a renowned software that has been in the spotlight for the last few years, as it allows users to easily create distributed storage environments meant either for commercial or research uses. This chapter gives a short description of what Hadoop is, and how it will be used in this thesis.

### 3.1 Overview

Hadoop, also known as Apache Hadoop, is an open-source framework initially released by Yahoo! and later developed by Apache Software Foundation, used for highly-scalable processing and storage of large data sets in a distributed manner.

This software allows users to build up clusters from any number of nodes in a common master-slave scheme, where the master is capable of individually handing out processing subtasks to the slaves and later building up the global result. This typical scenario, one of those Hadoop is often known for, is mostly carried out by one of its built-in modules called MapReduce. An illustration of the Hadoop engine and its modules is given below on Figure 3.1.

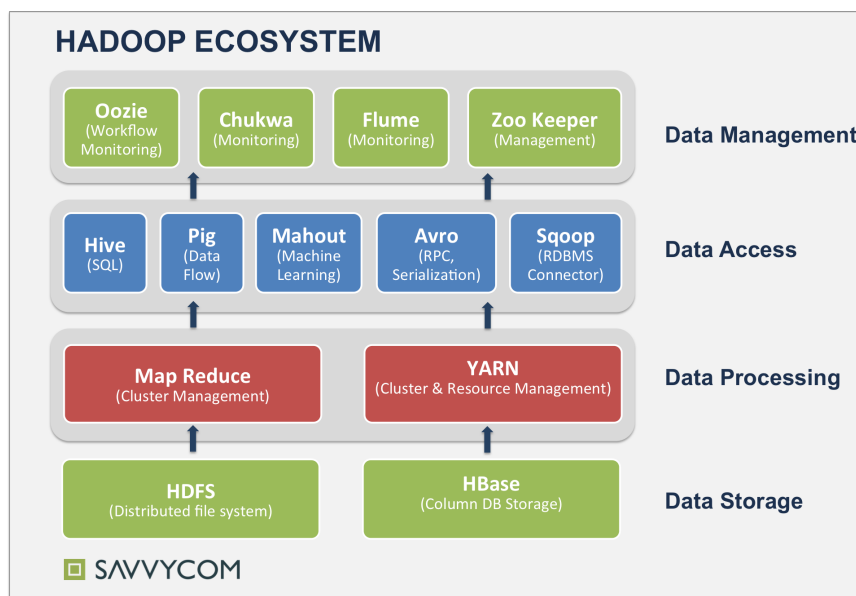


Figure 3.1: Hadoop engine and its modules [1].

One of the main features that characterizes Hadoop is its high fault-tolerance whenever the information is being treated or just stored. Every time a node from the cluster goes down, its file system will be reconstructed in a different node whenever possible while resuming any affected tasks. The reconstruction process is often achievable due to either a replication factor configured by the user, having several data copies found across the cluster, or an erasure code policy that allows the missing data chunks to be re-generated.

Additionally, Hadoop allows clusters to be increased in size at any time, even if nodes are already running, thus providing immense flexibility and scalability. This translates into huge computational power capabilities and the ability to process several terabytes of data in a short period of time.

## 3.2 Requirements

Procedures and results shown in this thesis correspond to Hadoop version 3.0.0-alpha2 running on Linux, which has a series of requirements:

- Java 5 or higher, preferably 6 or higher.
- SSH updated and configured, keys shared across the entire cluster thus allowing passwordless SSH access from/to each node.
- IPv6 disabled (recommended).
- Dedicated Hadoop user with admin privileges (recommended).

## 3.3 Configuration

Hadoop has a list of configuration files that define both functionality and behaviour for the cluster, mostly defining settings for the different modules (i.e., Yarn, MapReduce, etc.). All of these files can be found on the `$HADOOP_HOME/etc/hadoop/` folder, as shown in Figure 3.2.

Before modifying each file, it is important to update system variables on the `./bashrc` file (in this case `/etc/bashrc` on Fedora, or `/etc/profile` when changing settings permanently for all users). This enables Hadoop to execute commands from any directory without invoking the full installation path. The last few lines should look exactly like Figure 3.3, found below.

The next step is modifying the Java path on the Hadoop environment variables, found on `/etc/hadoop/hadoop-env.sh`. The only line that needs to be modified is shown in Figure 3.4.

The following files to modify, shown on Figures 3.5, 3.6, and 3.7 specify cluster behavior on the different Hadoop modules. File `/etc/hadoop/core-site.xml` defines key attributes regarding storage locations both locally and remotely within the cluster. Similarly, file `/etc/hadoop/mapred-site.xml` configures MapReduce folder paths, exclude list location for suspended nodes and module ports. Finally, `/etc/hadoop/hdfs-site.xml` determines key storage settings and is arguably the most important configuration file in Hadoop.

```

srl_sards@floeibanen.uib.nor-net $ pwd
/usr/local/hadoop/etc/hadoop
srl_sards@floeibanen.uib.nor-net $ ls
capacity-scheduler.xml      kms-site.xml
configuration.xml          log4j.properties
container-executor.cfg     mapred-env.cmd
core-site.xml              mapred-env.sh
hadoop-env.cmd            mapred-queues.xml.template
hadoop-env.sh             mapred-site.xml
hadoop-metrics2.properties mapred-site.xml.template
hadoop-policy.xml         masters
hadoop-user-functions.sh.example shellprofile.d
hdfs-site.xml             slaves
httpfs-env.sh            ssl-client.xml.example
httpfs-log4j.properties  ssl-server.xml.example
httpfs-signature.secret  workers
httpfs-site.xml          yarn-env.cmd
kms-acls.xml             yarn-env.sh
kms-env.sh              yarn-site.xml
kms-log4j.properties
srl_sards@floeibanen.uib.nor-net $ █

```

Figure 3.2: Hadoop configuration files.

```

export HADOOP_HOME=/usr/local/hadoop
export PATH=$PATH:$HADOOP_HOME/bin:$HADOOP_HOME/sbin
srl_sards@floeibanen.uib.nor-net $ █

```

Figure 3.3: Hadoop system variables.

```

# The java implementation to use. By default, this environment
# variable is REQUIRED on ALL platforms except OS X!
export JAVA_HOME=/usr/lib/jvm/jre-1.8.0-openjdk-1.8.0.121-8.b14.fc24.x86_64/

```

Figure 3.4: Hadoop environment variables.

```

<configuration>
  <property>
    <name>hadoop.tmp.dir</name>
    <value>/app/hadoop/tmp</value>
    <description>A base for other temporary directories.</description>
  </property>
  <property>
    <name>fs.default.name</name>
    <value>hdfs://master:9000</value>
    <description>The name of the default file system. A URI whose
    scheme and authority determine the FileSystem implementation. The
    uri's scheme determines the config property (fs.SCHEME.impl) naming
    the FileSystem implementation class. The uri's authority is used to
    determine the host, port, etc. for a filesystem.</description>
  </property>
  <property>
    <name>fs.trash.interval</name>
    <value>30</value>
  </property>
</configuration>
srl_sards@floeibanen.uib.nor-net $ █

```

Figure 3.5: File core-site.xml settings.



```

<configuration>
<property>
  <name>mapred.job.tracker</name>
  <value>master:54311</value>
  <description>The host and port that the MapReduce job tracker runs
  at. If "local", then jobs are run in-process as a single map
  and reduce task.
  </description>
</property>
<property>
  <name>mapred.local.dir</name>
  <value>/usr/local/hadoop/mapred/local</value>
</property>
<property>
  <name>mapred.temp.dir</name>
  <value>/usr/local/hadoop/mapred/temp</value>
</property>
<property>
  <name>mapred.hosts.exclude</name>
  <value>/usr/local/hadoop/etc/hadoop/mapredExclude</value>
</property>
</configuration>
srl_sards@floeibanen.uib.nor-net $ █

```

Figure 3.6: File mapred-site.xml settings.

```

<configuration>
<property>
  <name>dfs.replication</name>
  <value>1</value>
  <description>Default block replication.
  The actual number of replications can be specified when the file is created.
  The default is used if replication is not specified in create time.
  </description>
</property>
<property>
  <name>dfs.datanode.handler.count</name>
  <value>100</value>
</property>
<property>
  <name>dfs.image.transfer.timeout</name>
  <value>600000</value>
</property>
<property>
  <name>dfs.datanode.socket.write.timeout</name>
  <value>630000</value>
</property>
<property>
  <name>dfs.datanode.max.transfer.threads</name>
  <value>8192</value>
</property>
<property>
  <name>dfs.namenode.name.dir</name>
  <value>/usr/local/hadoop/namenode</value>
</property>
<property>
  <name>dfs.datanode.data.dir</name>
  <value>/usr/local/hadoop/datanode</value>
</property>
<property>
  <name>dfs.hosts.exclude</name>
  <value>/usr/local/hadoop/etc/hadoop/excludeList</value>
  <final>true</final>
</property>
</configuration>
srl_sards@floeibanen.uib.nor-net $ █

```

Figure 3.7: File hdfs-core.xml settings.

In order to allow the communication between each pair of nodes in the cluster, every server needs to have the complete list of IP-domain name pairs on their `/etc/hosts` file. Figure 3.8 specifies the `hosts` file content for the node that has been used as master during the measurements, namely `bakklandet.ntnu.nor-net`.

```

srl_sards@bakklandet.ntnu.nor-net $ cat /etc/hosts
127.0.0.1 localhost
10.1.9.101 master
10.1.9.101 bakklandet.ntnu.nor-net
10.1.9.102 byaasen.ntnu.nor-net
10.1.7.100 julenisse.uia.nor-net
10.1.5.103 sokn.uis.nor-net
10.1.5.104 rennesoey.uis.nor-net
10.1.5.105 fjoeloeu.uis.nor-net
10.1.5.100 mosteroey.uis.nor-net
10.1.5.101 klosteroeu.uis.nor-net
10.1.5.102 askje.uis.nor-net
10.1.7.104 skipsnisse.uia.nor-net
10.1.7.103 hagenisse.uia.nor-net
10.1.7.102 gaardsnisse.uia.nor-net
10.1.7.101 fjoesnisse.uia.nor-net
10.1.9.103 lerkendal.ntnu.nor-net
10.1.9.105 heimdalen.ntnu.nor-net
10.1.9.104 bymarka.ntnu.nor-net
10.1.9.100 bybro.ntnu.nor-net
10.1.4.105 kongsbakken.uit.nor-net
10.1.2.100 bjoervika.uio.nor-net
10.1.4.103 skarven.uit.nor-net
10.1.4.101 nansen.uit.nor-net
10.1.4.102 aunegaarden.uit.nor-net
10.1.4.104 arctandria.uit.nor-net
10.1.4.100 amundsen.uit.nor-net
srl_sards@bakklandet.ntnu.nor-net $

```

Figure 3.8: Hosts file on Master.

### 3.4 Adding New Erasure Coding Policies to Hadoop

In order to add new policies with different  $n$  and  $k$  parameters, but using one of the existing erasure coding policies in Hadoop (such as RS or XOR), three source files must be modified. Note that this short guide has been tested and used with Apache Hadoop version 3.0.0-alpha2, as different versions may provide slight changes in its file structure.

The first file to modify is named *ErasureCodeConstants.java*, and can be found under directory `/hadoop-common-project/hadoop-common/src/main/java/org/apache/hadoop/io/erasurecode/`, where a new policy needs to be specified as shown in Figure 3.9 for a new RS scheme with parameters  $n = 11$  and  $k = 8$ .

```

public static final String RS_DEFAULT_CODEC_NAME = "rs-default";
public static final String RS_LEGACY_CODEC_NAME = "rs-legacy";
public static final String XOR_CODEC_NAME = "xor";
public static final String HHXOR_CODEC_NAME = "hhxor";

public static final ECSchema RS_6_3_SCHEMA = new ECSchema(
    RS_DEFAULT_CODEC_NAME, 6, 3);

public static final ECSchema RS_3_2_SCHEMA = new ECSchema(
    RS_DEFAULT_CODEC_NAME, 3, 2);

public static final ECSchema RS_6_3_LEGACY_SCHEMA = new ECSchema(
    RS_LEGACY_CODEC_NAME, 6, 3);

public static final ECSchema XOR_2_1_SCHEMA = new ECSchema(
    XOR_CODEC_NAME, 2, 1);

public static final ECSchema RS_10_4_SCHEMA = new ECSchema(
    RS_DEFAULT_CODEC_NAME, 10, 4);

public static final ECSchema RS_8_3_SCHEMA = new ECSchema(
    RS_DEFAULT_CODEC_NAME, 8, 3);

```

Figure 3.9: Adding a new policy to Hadoop, step 1.

Subsequently, we need to add our policy to file *HdfsConstants.java*, found in location `/hadoop-hdfs-project/hadoop-hdfs-client/src/main/java/org/apache/hadoop/hdfs/protocol/`. This file is shown under Figure 3.10.

```
public static final byte RS_6_3_POLICY_ID = 0;
public static final byte RS_3_2_POLICY_ID = 1;
public static final byte RS_6_3_LEGACY_POLICY_ID = 2;
public static final byte XOR_2_1_POLICY_ID = 3;
public static final byte RS_10_4_POLICY_ID = 4;
public static final byte RS_8_3_POLICY_ID = 5;
```

Figure 3.10: Adding a new policy to Hadoop, step 2.

Finally, we need to include our scheme in file *ErasureCodingPolicyManager.java*, located under directory `hadoop-hdfs-project/hadoop-hdfs/src/main/java/org/apache/hadoop/hdfs/server/namenode/`. This will allow our policy to be listed when using command `hdfs erasurecode -listPolicies` once the cluster is running. This file is depicted below, in Figure 3.11.

```
private static final ErasureCodingPolicy SYS_POLICY1 =
    new ErasureCodingPolicy(ErasureCodeConstants.RS_6_3_SCHEMA,
        DEFAULT_CELL_SIZE, HdfsConstants.RS_6_3_POLICY_ID);
private static final ErasureCodingPolicy SYS_POLICY2 =
    new ErasureCodingPolicy(ErasureCodeConstants.RS_3_2_SCHEMA,
        DEFAULT_CELL_SIZE, HdfsConstants.RS_3_2_POLICY_ID);
private static final ErasureCodingPolicy SYS_POLICY3 =
    new ErasureCodingPolicy(ErasureCodeConstants.RS_6_3_LEGACY_SCHEMA,
        DEFAULT_CELL_SIZE, HdfsConstants.RS_6_3_LEGACY_POLICY_ID);
private static final ErasureCodingPolicy SYS_POLICY4 =
    new ErasureCodingPolicy(ErasureCodeConstants.XOR_2_1_SCHEMA,
        DEFAULT_CELL_SIZE, HdfsConstants.XOR_2_1_POLICY_ID);
private static final ErasureCodingPolicy SYS_POLICY5 =
    new ErasureCodingPolicy(ErasureCodeConstants.RS_10_4_SCHEMA,
        DEFAULT_CELL_SIZE, HdfsConstants.RS_10_4_POLICY_ID);
private static final ErasureCodingPolicy SYS_POLICY6 =
    new ErasureCodingPolicy(ErasureCodeConstants.RS_8_3_SCHEMA,
        DEFAULT_CELL_SIZE, HdfsConstants.RS_8_3_POLICY_ID);

//We may add more later.
private static final ErasureCodingPolicy[] SYS_POLICIES =
    new ErasureCodingPolicy[]{SYS_POLICY1, SYS_POLICY2, SYS_POLICY3,
        SYS_POLICY4, SYS_POLICY5, SYS_POLICY6};
```

Figure 3.11: Adding a new policy to Hadoop, step 3.

## 3.5 Recompiling Hadoop's Source Code

The Apache Hadoop project, described as an open-source project for reliable, scalable, and distributed computing, allows users to modify the code with relative ease. Source files can be downloaded directly from their github site<sup>1</sup> and locally extracted from the compressed *tar.gz* file. Release directory is often similar to the one shown on Figure 3.12.

<sup>1</sup>Apache Hadoop releases: <https://github.com/apache/hadoop/releases>.

```
maximiliano@maximiliano-VirtualBox:~/Downloads/hadoop-rel-release-3.0.0-alpha2$
ls
BUILDING.txt                hadoop-dist                 hadoop-tools
dev-support                 hadoop-hdfs-project        hadoop-yarn-project
hadoop-assemblies          hadoop-mapreduce-project  LICENSE.txt
hadoop-build-tools         hadoop-maven-plugins      NOTICE.txt
hadoop-client-modules      hadoop-minicluster        pom.xml
hadoop-cloud-storage-project hadoop-project             README.txt
hadoop-common-project      hadoop-project-dist       start-build-env.sh
```

Figure 3.12: Apache Hadoop release directory.

One of the main requirements for recompiling code is installing docker and all of its components, which must be available when re-building the package. The executable responsible for initializing the compiling process is *start-build-env.sh*, visualized on Figure 3.13.

```
maximiliano@5858ea679bed: ~/hadoop
---> 2b418117ec67
Step 4 : ENV HOME /home/maximiliano
---> Using cache
---> c8d5bcfb7bc2
Successfully built c8d5bcfb7bc2

Hadoop Dev

This is the standard Hadoop Developer build environment.
This has all the right tools installed required to build
Hadoop from source.

Low Memory

Your system is running on very little memory.
This means it may work but it wil most likely be slower than needed.

If you are running this via boot2docker you can simply increase
the available memory to atleast 2 GiB (you have 0 GiB )
maximiliano@5858ea679bed:~/hadoop$ mvn package -Pdist -DskipTests -Dtar
```

Figure 3.13: Apache Hadoop compiling executable.

Command outlined above will start the compiling process, which normally takes between 10-15 minutes, and in case there are no syntax errors present in the code, a successful compile message will be presented as seen in Figure 3.14.

```

maximiliano@5858ea679bed: ~/hadoop
[INFO] Apache Hadoop Archives ..... SUCCESS [3.247s]
[INFO] Apache Hadoop Archive Logs ..... SUCCESS [3.066s]
[INFO] Apache Hadoop Rumen ..... SUCCESS [6.688s]
[INFO] Apache Hadoop Gridmix ..... SUCCESS [5.428s]
[INFO] Apache Hadoop Data Join ..... SUCCESS [2.866s]
[INFO] Apache Hadoop Extras ..... SUCCESS [2.658s]
[INFO] Apache Hadoop Pipes ..... SUCCESS [0.092s]
[INFO] Apache Hadoop OpenStack support ..... SUCCESS [5.717s]
[INFO] Apache Hadoop Amazon Web Services support ..... SUCCESS [6.764s]
[INFO] Apache Hadoop Azure support ..... SUCCESS [5.572s]
[INFO] Apache Hadoop Aliyun OSS support ..... SUCCESS [3.805s]
[INFO] Apache Hadoop Client Aggregator ..... SUCCESS [4.925s]
[INFO] Apache Hadoop Mini-Cluster ..... SUCCESS [1.445s]
[INFO] Apache Hadoop Scheduler Load Simulator ..... SUCCESS [6.093s]
[INFO] Apache Hadoop Azure Data Lake support ..... SUCCESS [4.321s]
[INFO] Apache Hadoop Tools Dist ..... SUCCESS [4.594s]
[INFO] Apache Hadoop Kafka Library support ..... SUCCESS [2.935s]
[INFO] Apache Hadoop Tools ..... SUCCESS [0.100s]
[INFO] Apache Hadoop Client API ..... SUCCESS [1:11.314s]
[INFO] Apache Hadoop Client Runtime ..... SUCCESS [51.027s]
[INFO] Apache Hadoop Client Packaging Invariants ..... SUCCESS [0.646s]
[INFO] Apache Hadoop Client Test Minicluster ..... SUCCESS [1:41.665s]
[INFO] Apache Hadoop Client Packaging Invariants for Test ..... SUCCESS [0.443s]
[INFO] Apache Hadoop Client Packaging Integration Tests .. SUCCESS [0.431s]
[INFO] Apache Hadoop Distribution ..... SUCCESS [19.744s]
[INFO] Apache Hadoop Client Modules ..... SUCCESS [0.136s]
[INFO] Apache Hadoop Cloud Storage ..... SUCCESS [1.551s]
[INFO] Apache Hadoop Cloud Storage Project ..... SUCCESS [0.103s]
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 14:11.336s
[INFO] Finished at: Wed Nov 08 13:40:36 UTC 2017
[INFO] Final Memory: 134M/385M
[INFO] -----
maximiliano@5858ea679bed:~/hadoop$

```

Figure 3.14: Apache Hadoop compiling process.

After the re-compilation is complete, a compressed installation file will be located in folder `/hadoop-dist/target`, as depicted in Figure 3.15.

```

maximiliano@maximiliano-VirtualBox:~/Downloads/hadoop-rel-release-3.0.0-alpha2/h
adoop-dist/target$ ls
antrun          hadoop-3.0.0-alpha2.tar.gz      test-classes
classes        hadoop-tools-deps              test-dir
hadoop-3.0.0-alpha2  maven-shared-archive-resources

```

Figure 3.15: Apache Hadoop re-compiled file.

# Chapter 4

## NorNet Core

NorNet Core was introduced in 2012 by Simula Research Laboratory, as an effort to create a scenario where institutions and researchers can perform tests, experiments, analysis, and reviews regarding research projects they participate in. This chapter provides an overview of the NorNet environment, its contributing institutions, and how to access NorNet Core.

### 4.1 Description

NorNet Core, or NorNet Core TestBed [29] is a distributed academic network used for experimental networking research, initiated at Simula Research Laboratory in Norway. Although this network mainly consists of physical nodes located inside of Norway (see Figure 4.2), there are several nodes belonging to other partner institutions worldwide. The current list of participants is as follows:






















Country	Site	ISP 1	ISP 2	ISP 3	ISP 4
	Simula Research Laboratory	Uninett	Kvantel	Telenor	PowerTech
	Universitetet i Oslo	Uninett	Broadnet	PowerTech	
	Høgskolen i Gjøvik	Uninett	PowerTech		
	Universitetet i Tromsø	Uninett	Telenor	PowerTech	
	Universitetet i Stavanger	Uninett	Altibox	PowerTech	
	Universitetet i Bergen	Uninett	BKK		
	Universitetet i Agder	Uninett	PowerTech		
	Universitetet på Svalbard	Uninett	Telenor		
	Universitetet i Trondheim	Uninett	PowerTech		
	Høgskolen i Narvik	Uninett	Broadnet	PowerTech	
	Høgskolen i Oslo og Akershus	Uninett			
	Karlstads Universitetet	SUNET			
	Universität Kaiserslautern	SDFN			
	Hochschule Hamburg	DFN			
	Universität Duisburg-Essen	DFN	Versatel		
	Universität Darmstadt	DFN			
	Hainan University	CERNET	CnUnicom		
	Haikou College of Economics	CnTelecom	CERTNET		
	The University of Kansas	KanREN			
	Korea University	KREONET			
	National ICT Australia	AARNet			

Figure 4.1: NorNet Core current site list.



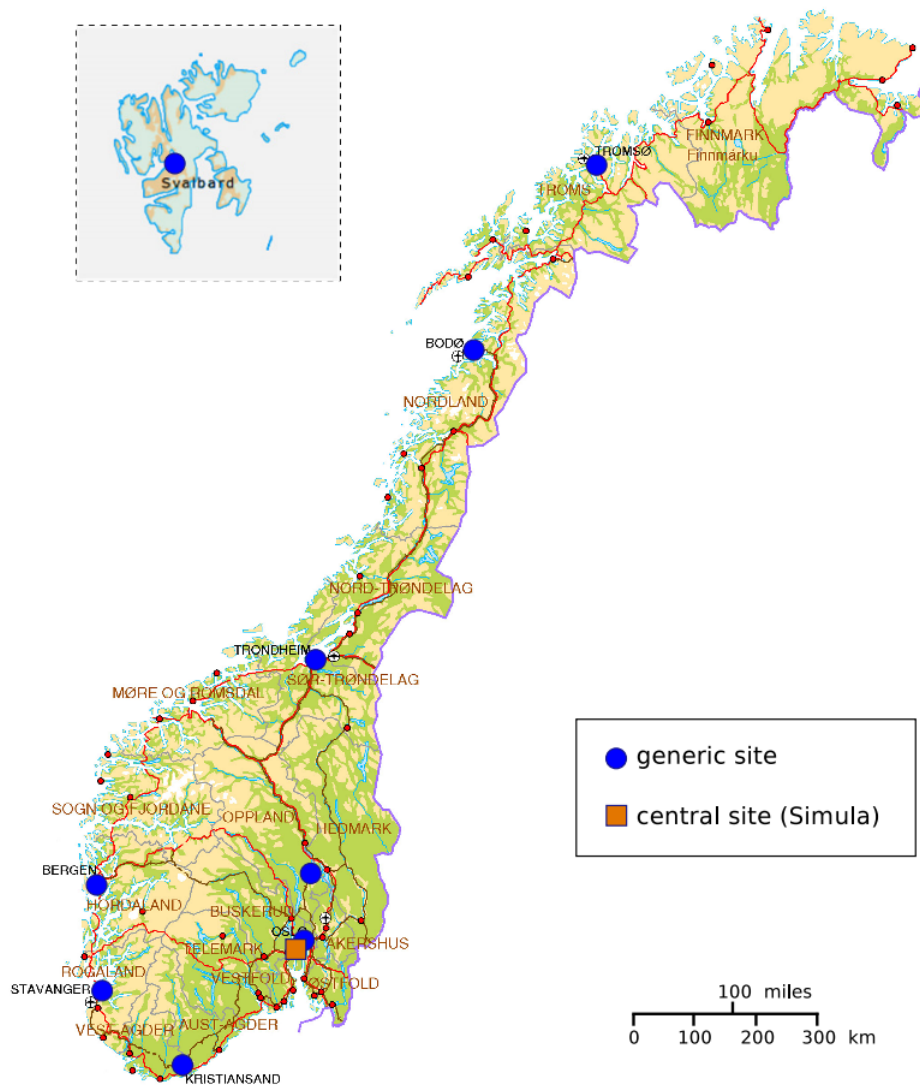


Figure 4.2: NorNet Core - Norwegian site locations.

As described in the NorNet Core Handbook [30], each site consists of one switch and four HP DL320 servers, all of which have exactly the same specifications (quad-core x64 CPUs, 8 GB RAM, 500 GB HD, and two 1000BASE-T Ethernet interfaces) and as of October 2017, each individual node runs Fedora 25.

The NorNet Core network also consists of an Ubuntu-based gatekeeper (specifically *gatekeeper.nntb.no*), the main point of access for users and the middle point between the physical nodes and the outer Internet.

## 4.2 Access

After transmitting their own private RSA keys to the gatekeeper on the user interface, users need to authenticate themselves through SSH. If the access is validated, the user is welcomed to the gatekeeper as shown on Figure 4.3.

```

maximiliano@oesthorn: ~
maximiliano@maximiliano-VirtualBox:~$ ssh maximiliano@gatekeeper.nntb.no
Welcome to Ubuntu 16.04.3 LTS (GNU/Linux 4.10.0-33-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/advantage

System information as of Tue Oct 17 16:49:49 CEST 2017

=====
# # ##### # # ##### #####
# # # # # # # # # # # #
# # # # # # # # # # ##### #
# # # # # ##### # # # # #
# ## # # # # # # # # #
# # ##### # # # # ##### #

-----
NorNet 1.2.5.1, packaged Thu Sep 7 15:29:59 CEST 2017
=====

Host:                oesthorn.simula.nornet
Uptime:              47 days
Used Memory:         8 x x86_64; 309 processes; 0 users
System:              Ubuntu 16.04 (xenial) with 4.10.0-33-generic
Load:                0.17, 0.04, 0.01
Used Memory:         158 MiB of 3821 MiB ( 265 MiB free)
Used Swap:           36 MiB of 4100 MiB ( 4064 MiB free)
Used Diskspace:     73% on /, 88% on /home
SSH Keys:
  SHA256:GkW8wEQPrewiLLkM6d0PKM5u3Bwm52VF9pNOYxkQkNY (DSA 1024)
  SHA256:RSM2TahZ6U0NYTNGYX0mVLTckJmbedX5r0WswJ+wmTo (ECDSA 256)
  SHA256:RXQwPJQ5Acxyyo4zChn12KyPWApKyI6qBCwu255m+2g (ED25519 256)
  6)
  SHA256:jUnyZFwmmWrgTC+X3B2M5E6p805K0LP17n99cNMxVs (RSA 2048)

Network:
  eth0:              10.1.1.6 10.2.1.6 10.4.1.6 10.9.1.6
                    2001:700:4100:101::6
                    2001:700:4100:201::6
                    2001:700:4100:401::6
                    2001:700:4100:901::6
  eth1:              128.39.37.188
                    2001:700:4100:2::188

0 packages can be updated.
0 updates are security updates.

*** System restart required ***
Last login: Tue Oct 17 16:49:16 2017 from 84.215.139.252
maximiliano@oesthorn:~$ █

```

Figure 4.3: Successful gatekeeper access.

Once users are successfully authenticated, they need to access each individual node through an additional SSH tunnel from the gatekeeper itself. Accesses often include the name of a specific slice, a sub-environment created on the gatekeeper meant to be used for a specific research project or group of projects. It is often necessary to specify the SSH private key location using the `-i` parameter on SSH. A typical access must be in the following form (as seen on Figure 4.4):

```

maximiliano@oesthorn:~$ ssh -i .ssh/gatekeeper srl_sards@kaiserberg.tukl.nor-net
Last login: Mon Oct 16 15:41:54 2017 from 2001:700:4100:101::6

System information as of Tue Oct 17 14:59:31 UTC 2017

=====
# # ##### # # ##### #####
# # # # # # # # # # # #
# # # # # # # # # # ##### #
# # # # # ##### # # # # #
# ## # # # # # # # # #
# # ##### # # # # ##### #

-----
Slice:                srl_sards (nor-net-f25-x86_64)
Host:                 kaiserberg.tukl.nor-net
Uptime:              50 days
Used Memory:         2 x x86_64; 9 processes; 0 users
System:              Fedora 25 (Twenty Five) with 4.4.84-888.fc25.x86_64
Load:                0.08, 0.02, 0.01
Used Memory:         20 MiB of 8796093022207 MiB (8796093021935 MiB free)
Used Swap:           207 MiB of 1023 MiB ( 816 MiB free)
Used Diskspace:     7% on /, 7% on /home
Network:
  eth0:              10.30.40.119
                    2001:700:4100:1e28::77:65
srl_sards@kaiserberg.tukl.nor-net $ █

```

Figure 4.4: Successful node access.

`ssh -i KEY/LOCATION SLICE_NAME@NODE_NAME . SITE . nor-net`



In the entirety of this project, the gatekeeper has been accessed from an Oracle VirtualBox VM running Ubuntu, created over a Windows local machine. The environment access order, as well as the operating system specification can be visualized on Figure 4.5.

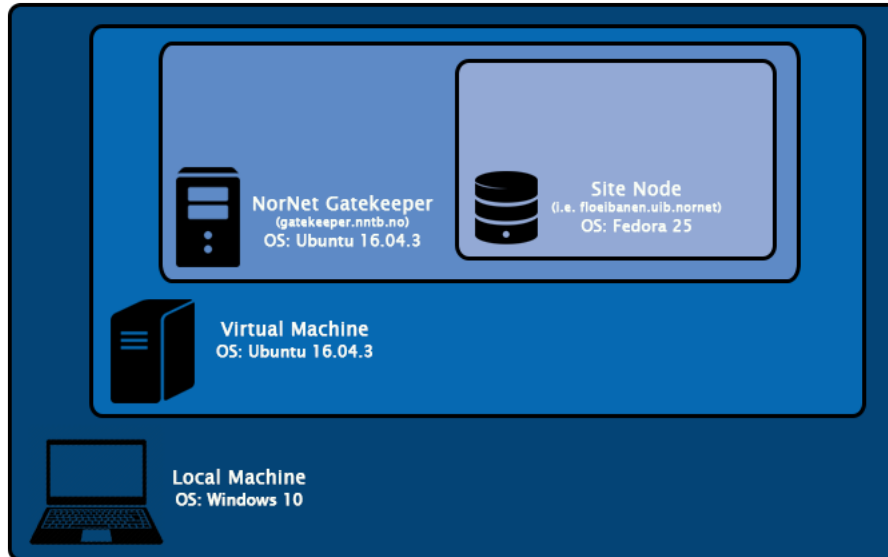


Figure 4.5: Operating systems and access order.

# Chapter 5

## Storage Simulations and Measurements

The following chapter describes the way results have been obtained in this thesis, including the software that has been used to assess network traffic and the list of NorNet Core nodes that intervened in our experiments.

### 5.1 Overall Concepts and Methodology Used

Across all simulations, clusters have been constrained only to Norwegian nodes in order to reduce latency to the minimum and standardize network bandwidth, thus increasing the comparability of the results. Since different sites might provide unequal bandwidth allowances, this could later on be reflected through disparate upload/download times between nodes during encoding and reconstruction, making them slightly biased.

Every simulation aims to upload and encode a file of approximate size 5 GB (more precisely, 5 473 128 572 bytes, or 5.097 GB) containing random content, and allow the reconstruction of one or many missing nodes through the interaction of the remaining participants. Cluster master will always be *bakklandet.ntnu.nor-net* (often found as *bakklandet*).

Two policies, considered the most relevant for our experiments, will be compared in this section. These are:

1. RS - RS(5, 3), RS(11, 8), RS(14, 10).
2. LRC (Pyramid) - PYR(6, 3), PYR(12, 8), PYR(15, 10).

Hadoop's default RS policies provided are RS(5, 3), RS(9, 6), and RS(14, 10). The RS(9, 6) policy will be discarded, as there exists no Pyramid counterpart with the same  $k$  in the industry to be compared with. On the other hand, PYR(12, 8) is a code introduced and described by Microsoft in 2007 [28], and seemed to be particularly relevant in this experiment. In order to allow the comparison with a corresponding RS code with the same  $k$ , an additional RS(11, 8) policy has been included. Although this scheme is not found within Hadoop's default policy list, it has been added manually through code injections.

Note that Pyramid policies used in this thesis need to be included in Hadoop's source code as they are not available by default. The construction process consists

in using two of the parity blocks to generate local parity groups, and using the remaining as global parities. A comparison of RS(5, 3) and PYR(6, 3) can be found below in Figure 5.1. Further details regarding our Pyramid code implementation are described in Section 6.2.

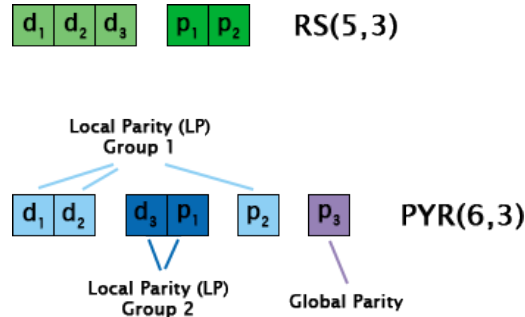


Figure 5.1: Pyramid code construction.

Finally, all of Hadoop’s encoding and decoding operations are handled under the finite field of order  $q = 256$ . They specifically provide a set of utility functions named *GF256.java*, which is called during most storage-related procedures. Even though this finite field is not strictly required when handling most of the codes analyzed in this thesis (and all RS codes), it enhances all byte operations and increases their efficiency.

## 5.2 Software Used

IPTraf-NG 1.1.4 is a console based network statistic monitoring utility that has been used to measure ongoing traffic between participating nodes in the cluster. Through a simple user interface (see Figure 5.2), it describes detailed information regarding network statistics such as IP, TCP, UDP, ICMP, non-IP and other IP packet counts, IP checksum errors, interface activity and packet size counts. It also allows users to generate traffic logs, which will be used thoroughly during simulations outlined in this thesis. A log sample is shown below in Figure 5.3.

```

iptraf-ng 1.1.4
TCP Connections (Source Host:Port) ----- Packets ----- Bytes ----- Flag ----- Iface -----
10.1.0.2:49731 > 292 15264 --A- tun0
192.168.75.1:222 > 294 134128 --PA- tun0
10.1.0.2:49738 > 10 3464 --A- tun0
192.168.75.1:800 > 10 2616 --PA- tun0
192.168.2.2:52361 > 13 2640 --A- red0
178.63.73.246:80 > 11 7014 --PA- red0
10.1.0.2:49739 > 4 1360 --A- tun0
192.168.75.1:800 > 4 700 --PA- tun0
10.1.0.2:49740 > 4 1335 --A- tun0
192.168.75.1:800 > 4 700 --PA- tun0
10.1.0.2:49735 > 4 1344 --A- tun0
192.168.75.1:800 > 4 700 --PA- tun0
10.1.0.2:49741 > 4 1335 --A- tun0
192.168.75.1:800 > 4 700 --PA- tun0
TCP: 30 entries ----- Active -----

UDP (129 bytes) from 192.168.110.2:57064 to 192.168.110.1:1197 on blue0
UDP (289 bytes) from 192.168.110.1:1197 to 192.168.110.2:57064 on blue0
UDP (129 bytes) from 192.168.110.2:57064 to 192.168.110.1:1197 on blue0
UDP (289 bytes) from 192.168.110.1:1197 to 192.168.110.2:57064 on blue0
UDP (129 bytes) from 192.168.110.2:57064 to 192.168.110.1:1197 on blue0
UDP (289 bytes) from 192.168.110.1:1197 to 192.168.110.2:57064 on blue0
UDP (129 bytes) from 192.168.110.2:57064 to 192.168.110.1:1197 on blue0

Bottom ----- Elapsed time: 0:00 -----
Packets captured: 2011 | TCP flow rate: 0.10 MBps
Up/Dn/PgUp/PgDn-scroll M-more TCP info W-chg actv win S-sort TCP X-exit

```

Figure 5.2: IPTraf-NG user interface.

```

Sun Oct 15 17:37:23 2017; TCP; eth0; 60 bytes; from fjoeloey.uis.nor-net:9866 to srl-sards.bakklandet.uninett.ntnu.nor-net:42333; FIN sent; 869 packets, 55512 bytes, avg flow rate 44.41 kbps
Sun Oct 15 17:37:23 2017; TCP; eth0; 60 bytes; from fjoeloey.uis.nor-net:9866 to srl-sards.bakklandet.uninett.ntnu.nor-net:55232; FIN sent; 2852 packets, 173133 bytes, avg flow rate 138.50 kbps
Sun Oct 15 17:37:23 2017; TCP; eth0; 60 bytes; from srl-sards.bakklandet.uninett.ntnu.nor-net:55232 to fjoeloey.uis.nor-net:9866; FIN acknowledged
Sun Oct 15 17:37:23 2017; TCP; eth0; 60 bytes; from srl-sards.bakklandet.uninett.ntnu.nor-net:55232 to fjoeloey.uis.nor-net:9866; FIN sent; 7874 packets, 11458572 bytes, avg flow rate 9166.86 kbps
Sun Oct 15 17:37:23 2017; TCP; eth0; 60 bytes; from fjoeloey.uis.nor-net:9866 to srl-sards.bakklandet.uninett.ntnu.nor-net:55232; FIN acknowledged
Sun Oct 15 17:37:23 2017; TCP; eth0; 60 bytes; from srl-sards.bakklandet.uninett.ntnu.nor-net:42333 to fjoeloey.uis.nor-net:9866; FIN acknowledged
Sun Oct 15 17:37:23 2017; TCP; eth0; 60 bytes; from srl-sards.bakklandet.uninett.ntnu.nor-net:42333 to fjoeloey.uis.nor-net:9866; FIN sent; 1683 packets, 2409618 bytes, avg flow rate 1927.69 kbps
Sun Oct 15 17:37:23 2017; TCP; eth0; 60 bytes; from fjoeloey.uis.nor-net:9866 to srl-sards.bakklandet.uninett.ntnu.nor-net:42333; FIN acknowledged
Sun Oct 15 17:37:23 2017; TCP; eth0; 60 bytes; from srl-sards.bakklandet.powertechnor-net:47584 to fjoeloey.uis.nor-net:9866; FIN acknowledged
Sun Oct 15 17:37:23 2017; TCP; eth0; 60 bytes; from srl-sards.bakklandet.powertechnor-net:47584 to fjoeloey.uis.nor-net:9866; FIN sent; 52 packets, 63520 bytes, avg flow rate 50.82 kbps
Sun Oct 15 17:37:23 2017; TCP; eth0; 60 bytes; from fjoeloey.uis.nor-net:9866 to srl-sards.bakklandet.powertech.ntnu.nor-net:47584; FIN acknowledged
Sun Oct 15 17:37:23 2017; TCP; eth0; 60 bytes; from srl-sards.bakklandet.powertechnor-net:42114 to fjoeloey.uis.nor-net:9866; FIN acknowledged
Sun Oct 15 17:37:23 2017; TCP; eth0; 60 bytes; from srl-sards.bakklandet.powertechnor-net:42114 to fjoeloey.uis.nor-net:9866; FIN sent; 65 packets, 63947 bytes, avg flow rate 51.15 kbps
Sun Oct 15 17:37:23 2017; TCP; eth0; 60 bytes; from fjoeloey.uis.nor-net:9866 to srl-sards.bakklandet.powertech.ntnu.nor-net:42114; FIN acknowledged
srl_sards@kaiserberg.tukl.nor-net $

```

Figure 5.3: Traffic log sample on a real cluster simulation.

### 5.3 Traffic Log-Harvesting Script

A simple script has been developed in order to gather network logs from each participating node, harvest only relevant information from them, calculate simulation totals and eliminate duplicates. Since a packet transferred from node  $c_1$  to  $c_2$  will appear on both nodes' logs (as ingoing and outgoing packets respectfully), they need to be canceled out as they may otherwise interfere in the simulation's measurement accuracy. This script also allows the user to specify start and end times, thus restricting the process to a specific time frame. Full source code can be found in Section 8.1.

Additionally, the code calls out for two text files (see Figures 5.4 and 5.5) that contain node DNS service name, as well as just node name which is the way they will be identified during the execution to filter log files.

```

srl_sards@kaiserberg.tukl.nor-net $ cat nodes
bakklandet.ntnu.nor-net
byaasen.ntnu.nor-net
heimdal.ntnu.nor-net
lerkendal.ntnu.nor-net
askje.uis.nor-net
fjoeloey.uis.nor-net
klosteroeuy.uis.nor-net
mosteroey.uis.nor-net
rennesoeuy.uis.nor-net
sokn.uis.nor-net
kongsbakken.uit.nor-net
skarven.uit.nor-net

```

Figure 5.4: Node DNS names file.

```

srl_sards@kaiserberg.tukl.nor-net $ cat node_names
bakklandet
byaasen
bybro
heimdal
lerkendal
askje
fjoeloey
master
klosteroeuy
mosteroey
rennesoeuy
sokn
kongsbakken
amundsen
aunegaarden
nansen
skarven
bjoervika

```

Figure 5.5: Node names file.

## 5.4 Participating Nodes

Nodes used for experimentation throughout this thesis work belong to the NorNet Core network and can be found on the list below. For the most part, nodes will be identified only by its name and not by their network site (i.e., *ntnu*, *uis*, *uit*) as they still are unique identifiers.

1. kaiserberg.tukl.nor-net (Log-Harvesting)
2. bakklandet.ntnu.nor-net (Master)
3. byaasen.ntnu.nor-net
4. bybro.ntnu.nor-net
5. heimdal.ntnu.nor-net
6. lerkendal.ntnu.nor-net
7. askje.uis.nor-net
8. fjoeloey.uis.nor-net
9. klosteroy.uis.nor-net
10. mosteroey.uis.nor-net
11. rennesoy.uis.nor-net
12. sokn.uis.nor-net
13. kongsbakken.uit.nor-net
14. amundsen.uit.nor-net
15. arctandria.uit.nor-net
16. aunegaarden.uit.nor-net
17. skarven.uit.nor-net
18. fjoesnise.uia.nor-net
19. bjoervika.uio.nor-net

# Chapter 6

## Results

The following chapter is composed of results obtained during our experimentation with Apache Hadoop and NorNet Core. They reflect how erasure coding policies compare to one another in terms of efficiency, repair bandwidth, duration, and fault-tolerance capabilities. The last section provides an overview of all the analyzed policies, and aims to summarize the pragmatic results achieved in this thesis.

### 6.1 RS Codes

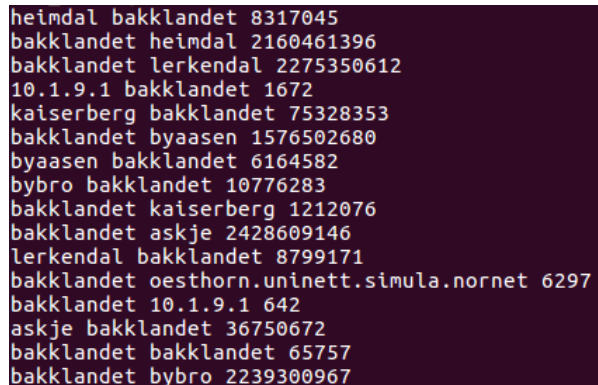
This section provides experiment results for the RS code policies on their different configurations. Both theory and literature suggest that reconstructions only require to contact  $k$  nodes, and that is also the case for Hadoop's recovery process. Although, network traffic measurements will more often than not contain more than  $k$  nodes transmitting information to the new node, as each stripe is re-generated from any random  $k$  nodes but these may vary on each cycle.

#### 6.1.1 RS(5,3)

Figure 6.2 shows the list of participating nodes and their content after the encoding phase, detailed in Figure 6.1. Note that each node contains 1.71 GB, which can otherwise be calculated as  $(5.097 \text{ GB} \times \frac{5}{3})/5$ .

When taking down one node before the reconstruction takes place, 14 blocks from this file structure will go missing as shown on Figure 6.3. After the reconstruction is completed (see Figures 6.4 and 6.5), all block groups will be shown as minimally erasure-coded, and under-replicated block messages will then disappear.

The reason behind blocks being *minimally* erasure-coded relies in the fact that during our experiments, a cluster of size  $n$  has been used to store the data, thus delivering each stripe to the entirety of the cluster.



```
heimdal bakklandet 8317045
bakklandet heimdal 2160461396
bakklandet lerkendal 2275350612
10.1.9.1 bakklandet 1672
kaiserberg bakklandet 75328353
bakklandet byaasen 1576502680
byaasen bakklandet 6164582
bybro bakklandet 10776283
bakklandet kaiserberg 1212076
bakklandet askje 2428609146
lerkendal bakklandet 8799171
bakklandet oesthorn.uninett.simula.nornet 6297
bakklandet 10.1.9.1 642
askje bakklandet 36750672
bakklandet bakklandet 65757
bakklandet bybro 2239300967
```

Figure 6.1: Traffic logs during encoding (in bytes) - RS,  $n = 5$ ,  $k = 3$ .

File encoding and the later uploading process has taken 11 minutes and 40 seconds, while reconstruction has had a 9 minutes and 6 seconds duration. As clearly seen on Figure 6.4, the later part of the process has taken place on node *fjoeloe.y.uis.nor.net*.

In operation

Show  entries Search:

Node	Http Address	Last contact	Capacity	Blocks	Block pool used	Version
✓ askje.uis.nor.net:9866 (10.1.5.217:9866)	askje.uis.nor.net:9864	2s	128.97 GB <div style="width: 100%;"></div>	14	1.71 GB (1.33%)	3.0.0-alpha2
✓ byaasen.ntnu.nor.net:9866 (10.1.9.198:9866)	byaasen.ntnu.nor.net:9864	2s	128.97 GB <div style="width: 100%;"></div>	14	1.71 GB (1.33%)	3.0.0-alpha2
✓ bybro.ntnu.nor.net:9866 (10.1.9.216:9866)	bybro.ntnu.nor.net:9864	2s	128.97 GB <div style="width: 100%;"></div>	14	1.71 GB (1.33%)	3.0.0-alpha2
✓ heimdalen.ntnu.nor.net:9866 (10.1.9.181:9866)	heimdal.ntnu.nor.net:9864	2s	128.97 GB <div style="width: 100%;"></div>	14	1.71 GB (1.33%)	3.0.0-alpha2
✓ lerkendal.ntnu.nor.net:9866 (10.1.9.217:9866)	lerkendal.ntnu.nor.net:9864	0s	128.97 GB <div style="width: 100%;"></div>	14	1.71 GB (1.33%)	3.0.0-alpha2

Showing 1 to 5 of 5 entries Previous **1** Next

Figure 6.2: Participating nodes - RS,  $n = 5$ ,  $k = 3$ .

```

FSCK started by dr.who (auth:SIMPLE) from /10.1.1.6 for path / at Sat Oct 14 15:27:15 UTC 2017
/RS_5_3/5GB: Under replicated BP-643533104-10.1.9.101-1507918838024:blk_-9223372036854775792_1001.
/RS_5_3/5GB: Under replicated BP-643533104-10.1.9.101-1507918838024:blk_-9223372036854775776_1002.
/RS_5_3/5GB: Under replicated BP-643533104-10.1.9.101-1507918838024:blk_-9223372036854775760_1003.
/RS_5_3/5GB: Under replicated BP-643533104-10.1.9.101-1507918838024:blk_-9223372036854775744_1004.
/RS_5_3/5GB: Under replicated BP-643533104-10.1.9.101-1507918838024:blk_-9223372036854775728_1005.
/RS_5_3/5GB: Under replicated BP-643533104-10.1.9.101-1507918838024:blk_-9223372036854775712_1006.
/RS_5_3/5GB: Under replicated BP-643533104-10.1.9.101-1507918838024:blk_-9223372036854775696_1007.
/RS_5_3/5GB: Under replicated BP-643533104-10.1.9.101-1507918838024:blk_-9223372036854775680_1008.
/RS_5_3/5GB: Under replicated BP-643533104-10.1.9.101-1507918838024:blk_-9223372036854775664_1009.
/RS_5_3/5GB: Under replicated BP-643533104-10.1.9.101-1507918838024:blk_-9223372036854775648_1010.
/RS_5_3/5GB: Under replicated BP-643533104-10.1.9.101-1507918838024:blk_-9223372036854775632_1011.
/RS_5_3/5GB: Under replicated BP-643533104-10.1.9.101-1507918838024:blk_-9223372036854775616_1012.
/RS_5_3/5GB: Under replicated BP-643533104-10.1.9.101-1507918838024:blk_-9223372036854775600_1013.
/RS_5_3/5GB: Under replicated BP-643533104-10.1.9.101-1507918838024:blk_-9223372036854775584_1014.

Status: HEALTHY
Number of data-nodes: 4
Number of racks:      1
Total dirs:           3
Total symlinks:       0

Replicated Blocks:
Total size: 0 B
Total files: 0
Total blocks (validated): 0
Minimally replicated blocks: 0
Over-replicated blocks: 0
Under-replicated blocks: 0
Mis-replicated blocks: 0
Default replication factor: 1
Average block replication: 0.0
Missing blocks: 0
Corrupt blocks: 0
Missing replicas: 0

Erasure Coded Block Groups:
Total size: 5473128672 B
Total files: 1
Total block groups (validated): 14 (avg. block group size 390937762 B)
Minimally erasure-coded block groups: 14 (100.0 %)
Over-erasure-coded block groups: 0 (0.0 %)
Under-erasure-coded block groups: 14 (100.0 %)
Unsatisfactory placement block groups: 0 (0.0 %)
Default ecPolicy: RS-DEFAULT-6-3-64k
Average block group size: 4.0
Missing block groups: 0
Corrupt block groups: 0
Missing internal blocks: 14 (20.0 %)
FSCK ended at Sat Oct 14 15:27:15 UTC 2017 in 15 milliseconds

The filesystem under path '/' is HEALTHY

```

Figure 6.3: Cluster status before reconstruction - RS,  $n = 5$ ,  $k = 3$ .

```
fjoeloy byaasen 15282630
heimdal fjoeloy 2231545837
byaasen fjoeloy 1627865523
fjoeloy fjoeloy 329225236
fjoeloy lerkendal 21272485
fjoeloy bybro 35225618
lerkendal fjoeloy 2828352914
bybro fjoeloy 2642831629
fjoeloy heimdal 22782611
```

Figure 6.4: Traffic logs during reconstruction (in bytes) - RS,  $n = 5$ ,  $k = 3$ .

```
FSCCK started by dr.who (auth:SIMPLE) from /10.1.1.6 for path / at Sat Oct 14 15:23:44 UTC 2017

Status: HEALTHY
Number of data-nodes: 5
Number of racks: 1
Total dirs: 3
Total symlinks: 0

Replicated Blocks:
Total size: 0 B
Total files: 0
Total blocks (validated): 0
Minimally replicated blocks: 0
Over-replicated blocks: 0
Under-replicated blocks: 0
Mis-replicated blocks: 0
Default replication factor: 1
Average block replication: 0.0
Missing blocks: 0
Corrupt blocks: 0
Missing replicas: 0

Erasure Coded Block Groups:
Total size: 5473128672 B
Total files: 1
Total block groups (validated): 14 (avg. block group size 390937762 B)
Minimally erasure-coded block groups: 14 (100.0 %)
Over-erasure-coded block groups: 0 (0.0 %)
Under-erasure-coded block groups: 0 (0.0 %)
Unsatisfactory placement block groups: 0 (0.0 %)
Default ecPolicy: RS-DEFAULT-6-3-64k
Average block group size: 5.0
Missing block groups: 0
Corrupt block groups: 0
Missing internal blocks: 0 (0.0 %)
FSCCK ended at Sat Oct 14 15:23:44 UTC 2017 in 22 milliseconds

The filesystem under path '/' is HEALTHY
```

Figure 6.5: Cluster status after reconstruction - RS,  $n = 5$ ,  $k = 3$ .

### 6.1.2 RS(11,8)

First figure found below (Figure 6.7) shows that each node contains approximately 650 MB (0.63 GB) after the encoding process (detailed in Figure 6.6 on the side), which is equivalent to  $(5.097 \text{ GB} \times \frac{11}{8})/11$ . This time 6 blocks will disappear from the file structure when one of the nodes goes down, as listed on top of Figure 6.8. File encoding lasted 8 minutes and 26 seconds, while reconstruction took 6 minutes and 10 seconds. Figure 6.9 shows the traffic details during reconstruction, where  $k = 8$  nodes transmitted their blocks to *skarven.uit.nornet*.

```
bakklandet heimdal 674202101
heimdal bakklandet 2705029
rennesoy bakklandet 12111787
master bakklandet 7897
bakklandet byaasen 651277012
bakklandet lerkendal 695300532
bakklandet sokn 669769487
bakklandet rennesoy 671187504
byaasen bakklandet 2599851
mosteroey bakklandet 12096406
bakklandet mosteroey 671732247
bakklandet askje 673570896
klosteroy bakklandet 11930653
amundsen bakklandet 10408629
bakklandet master 12563
bakklandet kongsbakken 670808487
trondheim.uninett.ntnu.nornet bybro 1254
bybro trondheim.uninett.ntnu.nornet 481
bakklandet fjoeloy 672109758
oesthorn.uninett.simula.nornet bakklandet 1824
lerkendal bakklandet 2789579
bakklandet amundsen 672372398
kongsbakken bakklandet 10721039
bakklandet oesthorn.uninett.simula.nornet 15903
fjoeloy bakklandet 11730063
bakklandet bakklandet 16302
askje bakklandet 11937258
bakklandet klosteroy 671241946
sokn bakklandet 12073767
```

Figure 6.6: Traffic logs during encoding (in bytes) - RS,  $n = 11$ ,  $k = 8$ .



## In operation

Show  entries Search:

Node	Http Address	Last contact	Capacity	Blocks	Block pool used	Version
✓ amundsen.uit.normet9866 (10.1.4.197:9866)	amundsen.uit.normet9864	0s	128.97 GB <div style="width: 100%;"></div>	6	693.21 MB (0.52%)	3.0.0-alpha2
✓ askje.uis.normet9866 (10.1.5.217:9866)	askje.uis.normet9864	1s	128.97 GB <div style="width: 100%;"></div>	6	689.12 MB (0.52%)	3.0.0-alpha2
✓ byaasen.ntnu.normet9866 (10.1.9.198:9866)	byaasen.ntnu.normet9864	0s	128.97 GB <div style="width: 100%;"></div>	6	671.7 MB (0.51%)	3.0.0-alpha2
✓ fjoeloeoy.uis.normet9866 (10.1.5.191:9866)	fjoeloeoy.uis.normet9864	0s	128.97 GB <div style="width: 100%;"></div>	6	663.66 MB (0.5%)	3.0.0-alpha2
✓ heimdal.ntnu.normet9866 (10.1.9.181:9866)	heimdal.ntnu.normet9864	2s	128.97 GB <div style="width: 100%;"></div>	6	783.32 MB (0.59%)	3.0.0-alpha2
✓ klosteroy.uis.normet9866 (10.1.5.215:9866)	klosteroy.uis.normet9864	0s	128.97 GB <div style="width: 100%;"></div>	6	778.9 MB (0.59%)	3.0.0-alpha2
✓ kongsbakken.uit.normet9866 (10.1.4.193:9866)	kongsbakken.uit.normet9864	2s	128.97 GB <div style="width: 100%;"></div>	6	763.91 MB (0.58%)	3.0.0-alpha2
✓ lerkendal.ntnu.normet9866 (10.1.9.217:9866)	lerkendal.ntnu.normet9864	1s	128.97 GB <div style="width: 100%;"></div>	6	700.65 MB (0.53%)	3.0.0-alpha2
✓ mosteroey.uis.normet9866 (10.1.5.197:9866)	mosteroey.uis.normet9864	0s	128.97 GB <div style="width: 100%;"></div>	6	688.75 MB (0.52%)	3.0.0-alpha2
✓ rennesoy.uis.normet9866 (10.1.5.216:9866)	rennesoy.uis.normet9864	1s	128.97 GB <div style="width: 100%;"></div>	6	681.13 MB (0.52%)	3.0.0-alpha2
✓ sokn.uis.normet9866 (10.1.5.194:9866)	sokn.uis.normet9864	1s	128.97 GB <div style="width: 100%;"></div>	6	727.63 MB (0.55%)	3.0.0-alpha2

Showing 1 to 11 of 11 entries Previous **1** Next

Figure 6.7: Participating nodes - RS,  $n = 11$ ,  $k = 8$ .

```

FSCK started by dr.who (auth:SIMPLE) from /10.1.1.6 for path / at Sun Oct 15 17:47:22 UTC 2017

/RS_11_8/5GB: Under replicated BP-828914993-10.1.9.101-1508088154902:blk_-9223372036854775792_1001.
/RS_11_8/5GB: Under replicated BP-828914993-10.1.9.101-1508088154902:blk_-9223372036854775776_1002.
/RS_11_8/5GB: Under replicated BP-828914993-10.1.9.101-1508088154902:blk_-9223372036854775760_1003.
/RS_11_8/5GB: Under replicated BP-828914993-10.1.9.101-1508088154902:blk_-9223372036854775744_1004.
/RS_11_8/5GB: Under replicated BP-828914993-10.1.9.101-1508088154902:blk_-9223372036854775728_1005.
/RS_11_8/5GB: Under replicated BP-828914993-10.1.9.101-1508088154902:blk_-9223372036854775712_1006.

Status: HEALTHY
Number of data-nodes: 11
Number of racks:      1
Total dirs:           2
Total symlinks:       0

Replicated Blocks:
Total size:           0 B
Total files:          0
Total blocks (validated): 0
Minimally replicated blocks: 0
Over-replicated blocks: 0
Under-replicated blocks: 0
Mis-replicated blocks: 0
Default replication factor: 1
Average block replication: 0.0
Missing blocks:       0
Corrupt blocks:       0
Missing replicas:     0

Erasure Coded Block Groups:
Total size:           5473128672 B
Total files:          1
Total block groups (validated): 6 (avg. block group size 912188112 B)
Minimally erasure-coded block groups: 6 (100.0 %)
Over-erasure-coded block groups: 0 (0.0 %)
Under-erasure-coded block groups: 6 (100.0 %)
Unsatisfactory placement block groups: 0 (0.0 %)
Default ecPolicy:     RS-DEFAULT-6-3-64k
Average block group size: 10.0
Missing block groups: 0
Corrupt block groups: 0
Missing internal blocks: 6 (9.090909 %)
FSCK ended at Sun Oct 15 17:47:22 UTC 2017 in 22 milliseconds

The filesystem under path '/' is HEALTHY

```

Figure 6.8: Cluster status before reconstruction - RS,  $n = 11$ ,  $k = 8$ .

```

heimdal skarven 833675428
askje skarven 915330308
mosteroey skarven 813726614
skarven kongsbakken 9423270
skarven byaasen 7305127
10.1.9.1 bakklandet 836
trondheim.uninett.ntnu.nornet bakklandet 4180
skarven skarven 322730566
lerkendal skarven 891964370
rennesoey skarven 449685501
klosteroey skarven 1209958880
bakklandet trondheim.uninett.ntnu.nornet 1605
skarven rennesoey 5799123
oesthorn.uninett.simula.nornet bakklandet 10670
kongsbakken skarven 485302651
skarven bybro 8295581
skarven mosteroey 12203938
bakklandet oesthorn.uninett.simula.nornet 25611
skarven klosteroey 14732546
byaasen skarven 749497372
skarven lerkendal 8378609
bybro skarven 834812171
skarven sokn 9589662
bakklandet 10.1.9.1 321
skarven heimdal 8856284
sokn skarven 593914165
skarven askje 11973372

```

Figure 6.9: Traffic logs during reconstruction (in bytes) - RS,  $n = 11$ ,  $k = 8$ .

```

FSCK started by dr.who (auth:SIMPLE) from /10.1.1.6 for path / at Sun Oct 15 14:46:41 UTC 2017

Status: HEALTHY
Number of data-nodes: 11
Number of racks: 1
Total dirs: 2
Total symlinks: 0

Replicated Blocks:
Total size: 0 B
Total files: 0
Total blocks (validated): 0
Minimally replicated blocks: 0
Over-replicated blocks: 0
Under-replicated blocks: 0
Mis-replicated blocks: 0
Default replication factor: 1
Average block replication: 0.0
Missing blocks: 0
Corrupt blocks: 0
Missing replicas: 0

Erasure Coded Block Groups:
Total size: 5473128672 B
Total files: 1
Total block groups (validated): 6 (avg. block group size 912188112 B)
Minimally erasure-coded block groups: 6 (100.0 %)
Over-erasure-coded block groups: 0 (0.0 %)
Under-erasure-coded block groups: 0 (0.0 %)
Unsatisfactory placement block groups: 0 (0.0 %)
Default ecPolicy: RS-DEFAULT-6-3-64k
Average block group size: 11.0
Missing block groups: 0
Corrupt block groups: 0
Missing internal blocks: 0 (0.0 %)
FSCK ended at Sun Oct 15 14:46:41 UTC 2017 in 7 milliseconds

The filesystem under path '/' is HEALTHY

```

Figure 6.10: Cluster status after reconstruction - RS,  $n = 11$ ,  $k = 8$ .

### 6.1.3 RS(14,10)

Figure 6.11 found below illustrates how after the encoding (detailed in Figure 6.12), each node stores files for approximately 526 MB (0.51 GB), otherwise equivalent to  $(5.097 \text{ GB} \times \frac{14}{10})/14$ . In this case 5 blocks will be excluded from the cluster storage when one of the participating nodes is turned off. This can be depicted on Figure 6.13. For this configuration the decoding process took 9 minutes and 14 seconds, while reconstruction surprisingly lasted only 2 minutes and 48 seconds. Figure 6.14 shows the network traffic during reconstruction, where nodes transmitted their contents to *bjoervika.wio.nornet*.

## In operation

Show  entries Search:

Node	Http Address	Last contact	Capacity	Blocks	Block pool used	Version
✓ amundsen.uit.nor-net:9866 (10.1.4.197:9866)	amundsen.uit.nor-net:9864	1s	128.97 GB <div style="width: 100%;"></div>	5	526.11 MB (0.4%)	3.0.0-alpha2
✓ askje.uis.nor-net:9866 (10.1.5.217:9866)	askje.uis.nor-net:9864	1s	128.97 GB <div style="width: 100%;"></div>	5	526.11 MB (0.4%)	3.0.0-alpha2
✓ aunegaarden.uit.nor-net:9866 (10.1.4.194:9866)	aunegaarden.uit.nor-net:9864	1s	128.97 GB <div style="width: 100%;"></div>	5	526.11 MB (0.4%)	3.0.0-alpha2
✓ bjoervika.uio.nor-net:9866 (10.1.2.196:9866)	bjoervika.uio.nor-net:9864	1s	128.97 GB <div style="width: 100%;"></div>	5	526.04 MB (0.4%)	3.0.0-alpha2
✓ byaasen.ntnu.nor-net:9866 (10.1.9.198:9866)	byaasen.ntnu.nor-net:9864	1s	128.97 GB <div style="width: 100%;"></div>	5	526.11 MB (0.4%)	3.0.0-alpha2
✓ fjoeloeys.uis.nor-net:9866 (10.1.5.191:9866)	fjoeloeys.uis.nor-net:9864	1s	128.97 GB <div style="width: 100%;"></div>	5	526.04 MB (0.4%)	3.0.0-alpha2
✓ heimdal.ntnu.nor-net:9866 (10.1.9.181:9866)	heimdal.ntnu.nor-net:9864	1s	128.97 GB <div style="width: 100%;"></div>	5	526.04 MB (0.4%)	3.0.0-alpha2
✓ klosteroy.uis.nor-net:9866 (10.1.5.215:9866)	klosteroy.uis.nor-net:9864	1s	128.97 GB <div style="width: 100%;"></div>	5	526.11 MB (0.4%)	3.0.0-alpha2
✓ kongsbakken.uit.nor-net:9866 (10.1.4.193:9866)	kongsbakken.uit.nor-net:9864	1s	128.97 GB <div style="width: 100%;"></div>	5	526.04 MB (0.4%)	3.0.0-alpha2
✓ lerkendal.ntnu.nor-net:9866 (10.1.9.217:9866)	lerkendal.ntnu.nor-net:9864	1s	128.97 GB <div style="width: 100%;"></div>	5	526.11 MB (0.4%)	3.0.0-alpha2
✓ mosteroey.uis.nor-net:9866 (10.1.5.197:9866)	mosteroey.uis.nor-net:9864	1s	128.97 GB <div style="width: 100%;"></div>	5	526.04 MB (0.4%)	3.0.0-alpha2
✓ rennesoey.uis.nor-net:9866 (10.1.5.216:9866)	rennesoey.uis.nor-net:9864	1s	128.97 GB <div style="width: 100%;"></div>	5	526.11 MB (0.4%)	3.0.0-alpha2
✓ skarven.uit.nor-net:9866 (10.1.4.181:9866)	skarven.uit.nor-net:9864	1s	128.97 GB <div style="width: 100%;"></div>	5	526.06 MB (0.4%)	3.0.0-alpha2
✓ sokn.uis.nor-net:9866 (10.1.5.194:9866)	sokn.uis.nor-net:9864	1s	128.97 GB <div style="width: 100%;"></div>	5	526.04 MB (0.4%)	3.0.0-alpha2

Showing 1 to 14 of 14 entries Previous **1** Next

Figure 6.11: Participating nodes - RS,  $n = 14$ ,  $k = 10$ .

```

bakklandet heimdal 540494036
heimdal bakklandet 2513838
rennesoey bakklandet 9608380
master bakklandet 13038
bakklandet byaasen 522255256
aunegaarden bakklandet 8975685
bakklandet lerkendal 514890516
trondheim.uninett.ntnu.nor-net bakklandet 1224
bakklandet sokn 546881089
bakklandet rennesoey 545186901
byaasen bakklandet 2498274
mosteroey bakklandet 9699814
bakklandet mosteroey 545503340
bakklandet askje 537921290
klosteroy bakklandet 9673310
bakklandet skarven 542374461
bakklandet aunegaarden 541075012
amundsen bakklandet 8542133
bakklandet master 15246
bakklandet kongsbakken 528772411
bakklandet trondheim.uninett.ntnu.nor-net 419
bakklandet fjoeloeys 543008757
oesthorn.uninett.simula.nor-net bakklandet 3152
fjoesnisse bakklandet 8187156
lerkendal bakklandet 2446141
bakklandet amundsen 539579064
kongsbakken bakklandet 6595531
bakklandet oesthorn.uninett.simula.nor-net 11214
fjoeloeys bakklandet 9513547
skarven bakklandet 8964471
bakklandet fjoesnisse 550317936
bakklandet bakklandet 14399
askje bakklandet 9558612
bakklandet klosteroy 540011880
sokn bakklandet 9730353

```

Figure 6.12: Traffic logs during encoding (in bytes) - RS,  $n = 14$ ,  $k = 10$ .

```

FSCK started by dr.who (auth:SIMPLE) from /10.1.1.6 for path / at Sun Oct 15 17:18:08 UTC 2017

/RS_14_10/5GB: Under replicated BP-2110932229-10.1.9.101-1508083082787:blk_-9223372036854775792_1001.
/RS_14_10/5GB: Under replicated BP-2110932229-10.1.9.101-1508083082787:blk_-9223372036854775776_1002.
/RS_14_10/5GB: Under replicated BP-2110932229-10.1.9.101-1508083082787:blk_-9223372036854775760_1003.
/RS_14_10/5GB: Under replicated BP-2110932229-10.1.9.101-1508083082787:blk_-9223372036854775744_1004.
/RS_14_10/5GB: Under replicated BP-2110932229-10.1.9.101-1508083082787:blk_-9223372036854775728_1005.

Status: HEALTHY
Number of data-nodes: 13
Number of racks: 1
Total dirs: 2
Total symlinks: 0

Replicated Blocks:
Total size: 0 B
Total files: 0
Total blocks (validated): 0
Minimally replicated blocks: 0
Over-replicated blocks: 0
Under-replicated blocks: 0
Mis-replicated blocks: 0
Default replication factor: 1
Average block replication: 0.0
Missing blocks: 0
Corrupt blocks: 0
Missing replicas: 0

Erasure Coded Block Groups:
Total size: 5473128672 B
Total files: 1
Total block groups (validated): 5 (avg. block group size 1094625734 B)
Minimally erasure-coded block groups: 5 (100.0 %)
Over-erasure-coded block groups: 0 (0.0 %)
Under-erasure-coded block groups: 5 (100.0 %)
Unsatisfactory placement block groups: 0 (0.0 %)
Default ecPolicy: RS-DEFAULT-6-3-64k
Average block group size: 13.0
Missing block groups: 0
Corrupt block groups: 0
Missing internal blocks: 5 (7.142857 %)
FSCK ended at Sun Oct 15 17:18:08 UTC 2017 in 42 milliseconds

The filesystem under path '/' is HEALTHY

```

Figure 6.13: Cluster status before reconstruction - RS,  $n = 14$ ,  $k = 10$ .

```

bjoervika fjoeloey 4074496
bjoervika bjoervika 656425319
byaasen bjoervika 377317700
trondheim.uninett.ntnu.nornet bakklandet 418
heimdal bjoervika 375832991
bjoervika skarven 1926922
fjoeloey bjoervika 362212194
skarven bjoervika 240384180
bjoervika lerkendal 3994922
bjoervika aunegaarden 1922492
bjoervika rennesoey 4196454
bjoervika kongsbakken 4282850
mosteroey bjoervika 361284860
bakklandet trondheim.uninett.ntnu.nornet 160
bjoervika mosteroey 3681998
bjoervika sokn 3072162
lerkendal bjoervika 503193403
oesthorn.uninett.simula.nornet bakklandet 992
aunegaarden bjoervika 190494712
rennesoey bjoervika 361328754
klosteroy bjoervika 470876470
bjoervika heimdal 3376096
bjoervika klosteroy 5104962
kongsbakken bjoervika 478585209
bjoervika byaasen 2875678
bjoervika askje 3569064
amundsen bjoervika 234213725
bakklandet oesthorn.uninett.simula.nornet 4876
bjoervika amundsen 1453204
askje bjoervika 354042796
sokn bjoervika 254117843

```

Figure 6.14: Traffic logs during reconstruction (in bytes) - RS,  $n = 14$ ,  $k = 10$ .

```

FSCK started by dr.who (auth:SIMPLE) from /10.1.1.6 for path / at Sun Oct 15 17:16:42 UTC 2017

Status: HEALTHY
Number of data-nodes: 14
Number of racks: 1
Total dirs: 2
Total symlinks: 0

Replicated Blocks:
Total size: 0 B
Total files: 0
Total blocks (validated): 0
Minimally replicated blocks: 0
Over-replicated blocks: 0
Under-replicated blocks: 0
Mis-replicated blocks: 0
Default replication factor: 1
Average block replication: 0.0
Missing blocks: 0
Corrupt blocks: 0
Missing replicas: 0

Erasure Coded Block Groups:
Total size: 5473128672 B
Total files: 1
Total block groups (validated): 5 (avg. block group size 1094625734 B)
Minimally erasure-coded block groups: 5 (100.0 %)
Over-erasure-coded block groups: 0 (0.0 %)
Under-erasure-coded block groups: 0 (0.0 %)
Unsatisfactory placement block groups: 0 (0.0 %)
Default ecPolicy: RS-DEFAULT-6-3-64k
Average block group size: 14.0
Missing block groups: 0
Corrupt block groups: 0
Missing internal blocks: 0 (0.0 %)
FSCK ended at Sun Oct 15 17:16:42 UTC 2017 in 18 milliseconds

The filesystem under path '/' is HEALTHY

```

Figure 6.15: Cluster status after reconstruction - RS,  $n = 14$ ,  $k = 10$ .

## 6.2 Pyramid Codes

This subsection shows results obtained while running a modified Pyramid-XOR policy, including Cauchy matrix alteration and coefficient-handling. Our code is based on the original XOR logic provided by Hadoop, and can be found under Section 8.4. This new policy aims to generate two local parity groups, reducing the bandwidth and minimum data required during recovery processes. On the other hand, duration should not be reduced as transmissions between nodes happen simultaneously, and the same amount of data per node is transferred.

When it comes to the generator matrix  $G$ , our objective is to replace the original version on the left (with  $n = 7$  and  $k = 4$ ) for a pyramid-oriented layout, on the right:

$$G = \begin{pmatrix} 1 & 0 & 0 & 0 & 71 & -89 & 12 \\ 0 & 1 & 0 & 0 & 122 & -70 & -89 \\ 0 & 0 & 1 & 0 & -89 & 71 & 35 \\ 0 & 0 & 0 & 1 & -70 & 122 & -114 \end{pmatrix} \rightarrow \begin{pmatrix} 1 & 0 & 0 & 0 & 71 & 0 & 12 \\ 0 & 1 & 0 & 0 & 122 & 0 & -89 \\ 0 & 0 & 1 & 0 & 0 & 71 & 35 \\ 0 & 0 & 0 & 1 & 0 & 122 & -114 \end{pmatrix}.$$

Similarly, the resulting parity check matrix  $H$  will be affected as shown below:

$$H = \begin{pmatrix} 71 & 122 & -89 & -70 & 1 & 0 & 0 \\ -89 & -70 & 71 & 122 & 0 & 1 & 0 \\ 12 & -89 & 35 & -144 & 0 & 0 & 1 \end{pmatrix} \rightarrow \begin{pmatrix} 71 & 122 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 71 & 122 & 0 & 1 & 0 \\ 12 & -89 & 35 & -144 & 0 & 0 & 1 \end{pmatrix}.$$

One important thing to mention is that Hadoop's functionality is far too complex to properly identify these matrix alterations, and will continue to transfer data

from  $k$  nodes during reconstruction regardless of these changes. For this reason, measurements will be adapted to match our pyramid encoding properties.

In order to analyze the traffic in the most rigorous way we will assign each node to a stripe index, assuming that each node contains all  $i$  indexes for each stripe. Then we will estimate an average between network traffic for recoveries on the first local parity group, second local parity group and global parities. For example, given the code in the second example (see 6.2.2 below), of length  $n = 12$  and dimension  $k = 8$ , we know for a fact that the first group involves transfers between 5 nodes, as well as the second group (but with different nodes), and the global parities make use of  $k$  random nodes to perform a recovery. Statistically we can claim that an average of these three individual measurements would give us an approximate amount of data required in average for a reconstruction, regardless of the block position missing in the stripe.

### 6.2.1 PYR(6,3)

Figure 6.17 shows the cluster status after the original file has been encoded and stored, detailing the network traffic in Figure 6.16. This process lasted 10 minutes and 33 seconds, where each node ended up containing a total amount of 1.71 GB ( $5.097 \text{ GB} \times \frac{6}{3}/6$ ).

After 14 blocks went missing, the reconstruction process began on recently joining node *fjoeloy.uis.nor-net* (see Figures 6.18 and 6.19). This time, the duration of the recovery (shown in Figure 6.20) has been 9 minutes and 32 seconds.

```

bakklandet heimdal 2277865369
heimdal bakklandet 8077371
master bakklandet 11581
bakklandet byaasen 1683038684
bakklandet lerkendal 2364906232
trondheim.uninett.ntnu.nor-net bakklandet 1784
byaasen bakklandet 6275480
bybro bakklandet 9091440
bakklandet askje 2383861671
bakklandet master 17936
bakklandet trondheim.uninett.ntnu.nor-net 642
bakklandet fjoeloy 2404746328
oesthorn.uninett.simula.nor-net bakklandet 6512
lerkendal bakklandet 8511467
bakklandet oesthorn.uninett.simula.nor-net 37170
fjoeloy bakklandet 41463025
bakklandet bakklandet 20517
askje bakklandet 41554770
bakklandet bybro 2387180545

```

Figure 6.16: Traffic logs during encoding (in bytes) - Pyramid,  $n = 6$ ,  $k = 3$ .

In operation

Show 25 entries Search:

Node	Http Address	Last contact	Capacity	Blocks	Block pool used	Version
askje.uis.nor-net:9866 (10.1.5.217:9866)	askje.uis.nor-net:9864	1s	128.97 GB <div style="width: 100%; height: 10px; background-color: #ccc; position: relative;"><div style="width: 100%; height: 10px; background-color: #28a745;"></div></div>	14	1.71 GB (1.33%)	3.0.0-alpha2
byaasen.ntnu.nor-net:9866 (10.1.9.198:9866)	byaasen.ntnu.nor-net:9864	0s	128.97 GB <div style="width: 100%; height: 10px; background-color: #ccc; position: relative;"><div style="width: 100%; height: 10px; background-color: #28a745;"></div></div>	14	1.71 GB (1.33%)	3.0.0-alpha2
bybro.ntnu.nor-net:9866 (10.1.9.216:9866)	bybro.ntnu.nor-net:9864	2s	128.97 GB <div style="width: 100%; height: 10px; background-color: #ccc; position: relative;"><div style="width: 100%; height: 10px; background-color: #28a745;"></div></div>	14	1.71 GB (1.33%)	3.0.0-alpha2
fjoeloy.uis.nor-net:9866 (10.1.5.191:9866)	fjoeloy.uis.nor-net:9864	0s	128.97 GB <div style="width: 100%; height: 10px; background-color: #ccc; position: relative;"><div style="width: 100%; height: 10px; background-color: #28a745;"></div></div>	14	1.71 GB (1.33%)	3.0.0-alpha2
heimdal.ntnu.nor-net:9866 (10.1.9.181:9866)	heimdal.ntnu.nor-net:9864	2s	128.97 GB <div style="width: 100%; height: 10px; background-color: #ccc; position: relative;"><div style="width: 100%; height: 10px; background-color: #28a745;"></div></div>	14	1.71 GB (1.33%)	3.0.0-alpha2
lerkendal.ntnu.nor-net:9866 (10.1.9.217:9866)	lerkendal.ntnu.nor-net:9864	1s	128.97 GB <div style="width: 100%; height: 10px; background-color: #ccc; position: relative;"><div style="width: 100%; height: 10px; background-color: #28a745;"></div></div>	14	1.71 GB (1.33%)	3.0.0-alpha2

Showing 1 to 6 of 6 entries Previous **1** Next

Figure 6.17: Participating nodes - Pyramid,  $n = 6$ ,  $k = 3$ .

```

FSCK started by dr.who (auth:SIMPLE) from /10.1.1.6 for path / at Sat Oct 28 00:52:07 UTC 2017
/PYR_6_3/5GB: Under replicated BP-1948686839-10.1.9.101-1509150657674:blk_-9223372036854775792_1001.
/PYR_6_3/5GB: Under replicated BP-1948686839-10.1.9.101-1509150657674:blk_-9223372036854775776_1002.
/PYR_6_3/5GB: Under replicated BP-1948686839-10.1.9.101-1509150657674:blk_-9223372036854775760_1003.
/PYR_6_3/5GB: Under replicated BP-1948686839-10.1.9.101-1509150657674:blk_-9223372036854775744_1004.
/PYR_6_3/5GB: Under replicated BP-1948686839-10.1.9.101-1509150657674:blk_-9223372036854775728_1005.
/PYR_6_3/5GB: Under replicated BP-1948686839-10.1.9.101-1509150657674:blk_-9223372036854775712_1006.
/PYR_6_3/5GB: Under replicated BP-1948686839-10.1.9.101-1509150657674:blk_-9223372036854775696_1007.
/PYR_6_3/5GB: Under replicated BP-1948686839-10.1.9.101-1509150657674:blk_-9223372036854775680_1008.
/PYR_6_3/5GB: Under replicated BP-1948686839-10.1.9.101-1509150657674:blk_-9223372036854775664_1009.
/PYR_6_3/5GB: Under replicated BP-1948686839-10.1.9.101-1509150657674:blk_-9223372036854775648_1010.
/PYR_6_3/5GB: Under replicated BP-1948686839-10.1.9.101-1509150657674:blk_-9223372036854775632_1011.
/PYR_6_3/5GB: Under replicated BP-1948686839-10.1.9.101-1509150657674:blk_-9223372036854775616_1012.
/PYR_6_3/5GB: Under replicated BP-1948686839-10.1.9.101-1509150657674:blk_-9223372036854775600_1013.
/PYR_6_3/5GB: Under replicated BP-1948686839-10.1.9.101-1509150657674:blk_-9223372036854775584_1014.

Status: HEALTHY
Number of data-nodes: 5
Number of racks: 1
Total dirs: 2
Total symlinks: 0

Replicated Blocks:
Total size: 0 B
Total files: 0
Total blocks (validated): 0
Minimally replicated blocks: 0
Over-replicated blocks: 0
Under-replicated blocks: 0
Mis-replicated blocks: 0
Default replication factor: 1
Average block replication: 0.0
Missing blocks: 0
Corrupt blocks: 0
Missing replicas: 0

Erasure Coded Block Groups:
Total size: 5473128672 B
Total files: 1
Total block groups (validated): 14 (avg. block group size 390937762 B)
Minimally erasure-coded block groups: 14 (100.0 %)
Over-erasure-coded block groups: 0 (0.0 %)
Under-erasure-coded block groups: 14 (100.0 %)
Unsatisfactory placement block groups: 0 (0.0 %)
Default ecPolicy: RS-DEFAULT-6-3-64k
Average block group size: 5.0
Missing block groups: 0
Corrupt block groups: 0
Missing internal blocks: 14 (16.666666 %)
FSCK ended at Sat Oct 28 00:52:07 UTC 2017 in 47 milliseconds

The filesystem under path '/' is HEALTHY

```

Figure 6.18: Cluster status before reconstruction - Pyramid,  $n = 6$ ,  $k = 3$ .

We can now start by calculating the total bandwidth output required to perform the most general reconstruction, using global parities. In this case, we could assign each participant node to a stripe index  $(c_1, c_2, \dots, c_6)$  as follows:

- $c_1 = \text{byaasen (LP1)}$
- $c_2 = \text{bybro (LP1)}$
- $c_3 = \text{heimdal (LP2)}$
- $c_4 = \text{lerkendal (LP2)}$
- $c_5 = \text{askje (LP1)}$
- $\langle \text{Missing node} \rangle$  (GP)
- $c_6 = \text{klosteroy (Recovery)}$ ,

```

klosteroy byaasen 16065703
heimdal klosteroy 2221185430
byaasen klosteroy 1448885836
klosteroy klosteroy 369205864
klosteroy lerkendal 31177405
klosteroy bybro 30126663
lerkendal klosteroy 2864554724
bybro klosteroy 2749238816
klosteroy heimdal 23673401
askje klosteroy 2673823521

```

Figure 6.19: Traffic logs during reconstruction (in bytes) - Pyramid,  $n = 6$ ,  $k = 3$ .



where LP stands for local parity and GP for global parity. Nodes tagged LP1 belong to the first local parity group, nodes tagged LP2 belong to the second local parity group, while nodes tagged GP store global parities. Since the traffic shown in Figure 6.19 already considers the transfers of  $k$  nodes per stripe, we will normalize them to  $n - 1 = 5$  (the number of nodes currently alive, and sending data over to the new node), multiplying each line by  $\frac{5}{3}$ , allowing us to properly calculate approximate traffic measurements for each local parity group later on.

We can now re-calculate an average of all incoming ( $t_i^{\text{in}}$ ) and outgoing ( $t_i^{\text{out}}$ ) traffic between all  $k$  combinations within our cluster ( $c_1, c_2, \dots, c_{n-1}$ ) leaving two nodes out on each round. Although this method may be slightly redundant, it will be re-obtained as a means to use retrieved measurements across all reconstructions in a consistent manner. We can otherwise refer to this calculation as:

$$\frac{\sum_{\forall G \subseteq \{1, \dots, n-1\}: |G|=k} \sum_{i \in G} (t_i^{\text{in}} + t_i^{\text{out}})}{\binom{n-1}{k}} = 12\,763\,159\,047 \text{ bytes (11.88 GB)}.$$

If instead we needed to do a reconstruction within the first local parity group, we could then assume that the missing node has either index 1, 2, or 5 and obtain an average amount of traffic sent and received from the group indexes. In the first case, we leave out index 1 (byaasen), and sum interactions between bybro, askje and klosteroe (the recovery node). For the second round, we take in byaasen, askje and klosteroe, and the process is repeated for every combination. The resulting measurement will in that case be:

$$\frac{\sum_{g \in G_1 = \{1, 2, 5\}} \sum_{i \in G_1 \setminus \{g\}} (t_i^{\text{in}} + t_i^{\text{out}})}{|G_1|} = 8\,302\,165\,928 \text{ bytes (7.73 GB)}.$$

For the second local parity group, since it is too small to be encoded (having a size of 2) we can simply calculate the required traffic as a raw transfer from the remaining node in the group, as data should be an exact copy. This can be formalized as:

$$\frac{\sum_{g \in G_2 = \{3, 4\}} \sum_{i \in G_2 \setminus \{g\}} (t_i^{\text{in}} + t_i^{\text{out}})}{|G_2|} = 4\,899\,168\,907 \text{ bytes (4.56 GB)}.$$

Finally, as we have three indexes that can be repaired through the first local parity group, two by the second group, and only one that require involving global coefficients, we can estimate the required traffic using this policy as:

$$\begin{aligned} & \frac{3 \times 8\,302\,165\,928 + 2 \times 4\,899\,168\,907 + 12\,763\,159\,047}{6} \\ & = \mathbf{7\,911\,332\,440 \text{ bytes (7.37 GB)}}. \end{aligned}$$



```

FSCK started by dr.who (auth:SIMPLE) from /10.1.1.6 for path / at Sat Oct 28 00:47:12 UTC 2017

Status: HEALTHY
Number of data-nodes: 6
Number of racks: 1
Total dirs: 2
Total symlinks: 0

Replicated Blocks:
Total size: 0 B
Total files: 0
Total blocks (validated): 0
Minimally replicated blocks: 0
Over-replicated blocks: 0
Under-replicated blocks: 0
Mis-replicated blocks: 0
Default replication factor: 1
Average block replication: 0.0
Missing blocks: 0
Corrupt blocks: 0
Missing replicas: 0

Erasure Coded Block Groups:
Total size: 5473128672 B
Total files: 1
Total block groups (validated): 14 (avg. block group size 390937762 B)
Minimally erasure-coded block groups: 14 (100.0 %)
Over-erasure-coded block groups: 0 (0.0 %)
Under-erasure-coded block groups: 0 (0.0 %)
Unsatisfactory placement block groups: 0 (0.0 %)
Default ecPolicy: RS-DEFAULT-6-3-64k
Average block group size: 6.0
Missing block groups: 0
Corrupt block groups: 0
Missing internal blocks: 0 (0.0 %)
FSCK ended at Sat Oct 28 00:47:12 UTC 2017 in 15 milliseconds

The filesystem under path '/' is HEALTHY

```

Figure 6.20: Cluster status after reconstruction - Pyramid,  $n = 6$ ,  $k = 3$ .

## 6.2.2 PYR(12,8)

In this scheme, Figure 6.22 shows the initial node status before the reconstruction. Each node contains a total amount of 657.6 MB ( $5.097 \text{ GB} \times \frac{12}{8} / 12$ ).

File encoding and upload has taken 8 minutes and 23 seconds, whereas node recovery towards skarven.uit.nor-net has had a 5 minutes and 6 seconds duration. Network traffic monitored during these processes can be found on Figures 6.21 (to the side) and 6.24.

```

bakklandet heimdal 912417432
heimdal bakklandet 3189367
rennesoey bakklandet 17255650
master bakklandet 10771
bakklandet byaasen 671486965
aunegaarden bakklandet 15131653
bakklandet lerkendal 913248624
trondheim.uninett.ntnu.nor-net bakklandet 3344
bakklandet sokn 664604063
bakklandet rennesoey 982000195
byaasen bakklandet 2486244
mosteroey bakklandet 11747734
bybro bakklandet 5154626
bakklandet mosteroey 673492933
bakklandet askje 964433875
klosteroey bakklandet 17368955
bakklandet aunegaarden 958769266
amundsen bakklandet 10359862
bakklandet master 15584
bakklandet kongsbakken 540122807
bakklandet trondheim.uninett.ntnu.nor-net 1284
oesthorn.uninett.simula.nor-net bakklandet 12209
lerkendal bakklandet 3325420
bakklandet amundsen 661197463
kongsbakken bakklandet 8387041
bakklandet oesthorn.uninett.simula.nor-net 42956
askje bakklandet 17121719
bakklandet klosteroey 969362199
bakklandet bakklandet 19093
sokn bakklandet 11900691
bakklandet bybro 895543308

```

Figure 6.21: Traffic logs during encoding (in bytes) - Pyramid,  $n = 12$ ,  $k = 8$ .

## In operation

Show  entries Search:

Node	Http Address	Last contact	Capacity	Blocks	Block pool used	Version
✓ amundsen.uit.normet9866 (10.1.4.197:9866)	amundsen.uit.normet9864	0s	128.97 GB <div style="width: 100%;"></div>	6	657.63 MB (0.5%)	3.0.0-alpha2
✓ askje.uis.normet9866 (10.1.5.217:9866)	askje.uis.normet9864	0s	128.97 GB <div style="width: 100%;"></div>	6	657.63 MB (0.5%)	3.0.0-alpha2
✓ aunegaarden.uit.normet9866 (10.1.4.194:9866)	aunegaarden.uit.normet9864	2s	128.97 GB <div style="width: 100%;"></div>	6	657.56 MB (0.5%)	3.0.0-alpha2
✓ byaasen.ntnu.normet9866 (10.1.9.198:9866)	byaasen.ntnu.normet9864	0s	128.97 GB <div style="width: 100%;"></div>	6	657.56 MB (0.5%)	3.0.0-alpha2
✓ bybro.ntnu.normet9866 (10.1.9.216:9866)	bybro.ntnu.normet9864	0s	128.97 GB <div style="width: 100%;"></div>	6	657.56 MB (0.5%)	3.0.0-alpha2
✓ heimdal.ntnu.normet9866 (10.1.9.181:9866)	heimdal.ntnu.normet9864	0s	128.97 GB <div style="width: 100%;"></div>	6	657.63 MB (0.5%)	3.0.0-alpha2
✓ klosteroy.uis.normet9866 (10.1.5.215:9866)	klosteroy.uis.normet9864	0s	128.97 GB <div style="width: 100%;"></div>	6	657.59 MB (0.5%)	3.0.0-alpha2
✓ kongsbakken.uit.normet9866 (10.1.4.193:9866)	kongsbakken.uit.normet9864	0s	128.97 GB <div style="width: 100%;"></div>	6	657.56 MB (0.5%)	3.0.0-alpha2
✓ lerkendal.ntnu.normet9866 (10.1.9.217:9866)	lerkendal.ntnu.normet9864	0s	128.97 GB <div style="width: 100%;"></div>	6	657.63 MB (0.5%)	3.0.0-alpha2
✓ mosteroey.uis.normet9866 (10.1.5.197:9866)	mosteroey.uis.normet9864	2s	128.97 GB <div style="width: 100%;"></div>	6	657.63 MB (0.5%)	3.0.0-alpha2
✓ rennesoy.uis.normet9866 (10.1.5.216:9866)	rennesoy.uis.normet9864	0s	128.97 GB <div style="width: 100%;"></div>	6	657.56 MB (0.5%)	3.0.0-alpha2
✓ sokn.uis.normet9866 (10.1.5.194:9866)	sokn.uis.normet9864	0s	128.97 GB <div style="width: 100%;"></div>	6	657.56 MB (0.5%)	3.0.0-alpha2

Showing 1 to 12 of 12 entries Previous **1** Next

Figure 6.22: Participating nodes - Pyramid,  $n = 12$ ,  $k = 8$ .

```

FSCK started by dr.who (auth:SIMPLE) from /10.1.1.6 for path / at Sun Nov 05 21:01:48 UTC 2017

/PYR_12_8/5GB: Under replicated BP-817701755-10.1.9.101-1509912902031:blk_-9223372036854775792_1001.
/PYR_12_8/5GB: Under replicated BP-817701755-10.1.9.101-1509912902031:blk_-9223372036854775776_1002.
/PYR_12_8/5GB: Under replicated BP-817701755-10.1.9.101-1509912902031:blk_-9223372036854775760_1003.
/PYR_12_8/5GB: Under replicated BP-817701755-10.1.9.101-1509912902031:blk_-9223372036854775744_1004.
/PYR_12_8/5GB: Under replicated BP-817701755-10.1.9.101-1509912902031:blk_-9223372036854775728_1005.
/PYR_12_8/5GB: Under replicated BP-817701755-10.1.9.101-1509912902031:blk_-9223372036854775712_1006.

Status: HEALTHY
Number of data-nodes: 11
Number of racks:      1
Total dirs:           2
Total symlinks:       0

Replicated Blocks:
Total size:           0 B
Total files:          0
Total blocks (validated): 0
Minimally replicated blocks: 0
Over-replicated blocks: 0
Under-replicated blocks: 0
Mis-replicated blocks: 0
Default replication factor: 1
Average block replication: 0.0
Missing blocks:       0
Corrupt blocks:       0
Missing replicas:     0

Erasure Coded Block Groups:
Total size:           5473128672 B
Total files:          1
Total block groups (validated): 6 (avg. block group size 912188112 B)
Minimally erasure-coded block groups: 6 (100.0 %)
Over-erasure-coded block groups: 0 (0.0 %)
Under-erasure-coded block groups: 6 (100.0 %)
Unsatisfactory placement block groups: 0 (0.0 %)
Default ecPolicy:    RS-DEFAULT-6-3-64k
Average block group size: 11.0
Missing block groups: 0
Corrupt block groups: 0
Missing internal blocks: 6 (8.333333 %)
FSCK ended at Sun Nov 05 21:01:48 UTC 2017 in 21 milliseconds

The filesystem under path '/' is HEALTHY

```

Figure 6.23: Cluster status before reconstruction - Pyramid,  $n = 12$ ,  $k = 8$ .

Similarly to previous cases, we can now analyze reconstructions using global parities. Allocating each node to a stripe position, we can identify our cluster as:

- $c_1 = \text{byaasen (LP1)}$
- $c_2 = \text{bybro (LP1)}$
- $c_3 = \text{heimdal (LP1)}$
- $c_4 = \text{lerkendal (LP1)}$
- $c_5 = \text{askje (LP2)}$
- $c_6 = \text{klosteroeey (LP2)}$
- $c_7 = \text{mosteroey (LP2)}$
- $c_8 = \text{rennesoey (LP2)}$
- $c_9 = \text{sokn (LP2)}$
- $c_{10} = \text{kongsbakken (LP1)}$
- $c_{11} = \text{amundsen (GP)}$
- $\langle \text{Missing node} \rangle \text{ (GP)}$
- $c_{12} = \text{skarven (Recovery)}$ .

```
heimdal skarven 1174971451
mosteroey skarven 451651622
askje skarven 933449951
skarven kongsbakken 14361436
skarven byaasen 6201551
skarven skarven 527343380
lerkendal skarven 771307311
rennesoey skarven 682297922
amundsen skarven 498262481
klosteroeey skarven 745952634
skarven rennesoey 6874506
kongsbakken skarven 865995886
skarven bybro 9773293
skarven mosteroey 4349817
skarven klosteroeey 6995924
byaasen skarven 620736213
skarven lerkendal 5598089
bybro skarven 830342614
skarven sokn 3829711
skarven amundsen 8203779
skarven heimdal 11835727
sokn skarven 347735486
skarven askje 9161921
```

Figure 6.24: Traffic logs during reconstruction (in bytes) - Pyramid,  $n = 12$ ,  $k = 8$ .

To make this global reconstruction possible we require transfers from  $k = 8$  nodes, and after normalizing our results by  $\frac{11}{8}$ , we can obtain the average in-/outgoing network traffic as:

$$\frac{\sum_{\forall G \subseteq \{1, \dots, n-1\}: |G|=k} \sum_{i \in G} (t_i^{\text{in}} + t_i^{\text{out}})}{\binom{n-1}{k}} = 8\,704\,001\,288 \text{ bytes (8.11 GB)}.$$

If instead we needed to do a reconstruction within the first local parity group, we can assume the missing node is either  $c_1, c_2, c_3, c_4$  or  $c_{10}$ . We then leave out one of these nodes at a time, counting all the interactions between the rest of the nodes and  $c_{12}$  (our recovery node), and obtain the average. This can be defined as:

$$\frac{\sum_{g \in G_1 = \{1, 2, 3, 4, 10\}} \sum_{i \in G_1 \setminus \{g\}} (t_i^{\text{in}} + t_i^{\text{out}})}{|G_1|} = 5\,467\,333\,076 \text{ bytes (5.09 GB)}.$$

Similarly, for the second local parity group we can obtain it as:

$$\frac{\sum_{g \in G_2 = \{5, 6, 7, 8, 9\}} \sum_{i \in G_2 \setminus \{g\}} (t_i^{\text{in}} + t_i^{\text{out}})}{|G_2|} = 4\,236\,626\,591 \text{ bytes (3.95 GB)}.$$

Subsequently, knowing that for this policy there are two local parity groups of size 5, as well as two global parities, we can generalize our traffic measurements for all cases as:

$$\frac{5 \times 5\,467\,333\,076 + 5 \times 4\,236\,626\,591 + 2 \times 8\,704\,001\,288}{12}$$

12

**= 5 493 983 409 bytes (5.1 GB).**

```

FSCK started by dr.who (auth:SIMPLE) from /10.1.1.6 for path / at Sun Nov 05 21:14:39 UTC 2017

Status: HEALTHY
Number of data-nodes: 12
Number of racks:      1
Total dirs:           2
Total symlinks:       0

Replicated Blocks:
Total size: 0 B
Total files: 0
Total blocks (validated): 0
Minimally replicated blocks: 0
Over-replicated blocks: 0
Under-replicated blocks: 0
Mis-replicated blocks: 0
Default replication factor: 1
Average block replication: 0.0
Missing blocks: 0
Corrupt blocks: 0
Missing replicas: 0

Erasure Coded Block Groups:
Total size: 5473128672 B
Total files: 1
Total block groups (validated): 6 (avg. block group size 912188112 B)
Minimally erasure-coded block groups: 6 (100.0 %)
Over-erasure-coded block groups: 0 (0.0 %)
Under-erasure-coded block groups: 0 (0.0 %)
Unsatisfactory placement block groups: 0 (0.0 %)
Default ecPolicy: RS-DEFAULT-6-3-64k
Average block group size: 12.0
Missing block groups: 0
Corrupt block groups: 0
Missing internal blocks: 0 (0.0 %)
FSCK ended at Sun Nov 05 21:14:39 UTC 2017 in 18 milliseconds

The filesystem under path '/' is HEALTHY

```

Figure 6.25: Cluster status after reconstruction - Pyramid,  $n = 12$ ,  $k = 8$ .

### 6.2.3 PYR(15,10)

Figure 6.26 depicts the content on each participating node after the file encoding, which can otherwise be obtained as  $(5.097\text{ GB} \times \frac{15}{10})/15$ . Encoding allocation lasted 9 minutes and 18 seconds, while recovery lasted for 4 minutes and 31 seconds. Figure 6.29 shows the traffic details during reconstruction, where  $k$  nodes transmitted their blocks to *bjoervika.uio.nor-net*.

## In operation

Show  entries Search:

Node	Http Address	Last contact	Capacity	Blocks	Block pool used	Version
✓ amundsens.uit.nor-net-9866 (10.1.4.197:9866)	amundsens.uit.nor-net-9866	1s	128.97 GB	5	526.04 MB (0.4%)	3.0.0-alpha2
✓ askje.uis.nor-net-9866 (10.1.5.217:9866)	askje.uis.nor-net-9866	2s	128.97 GB	5	526.06 MB (0.4%)	3.0.0-alpha2
✓ aunegaarden.uit.nor-net-9866 (10.1.4.194:9866)	aunegaarden.uit.nor-net-9866	2s	128.97 GB	5	526.11 MB (0.4%)	3.0.0-alpha2
✓ byaasen.ntnu.nor-net-9866 (10.1.9.198:9866)	byaasen.ntnu.nor-net-9866	0s	128.97 GB	5	526.11 MB (0.4%)	3.0.0-alpha2
✓ bybro.ntnu.nor-net-9866 (10.1.9.216:9866)	bybro.ntnu.nor-net-9866	2s	128.97 GB	5	526.11 MB (0.4%)	3.0.0-alpha2
✓ fjoeloey.uis.nor-net-9866 (10.1.5.191:9866)	fjoeloey.uis.nor-net-9866	0s	128.97 GB	5	526.04 MB (0.4%)	3.0.0-alpha2
✓ fjoesnisse.uis.nor-net-9866 (10.1.7.189:9866)	fjoesnisse.uis.nor-net-9866	41s	128.97 GB	5	526.11 MB (0.4%)	3.0.0-alpha2
✓ heimdal.ntnu.nor-net-9866 (10.1.9.181:9866)	heimdal.ntnu.nor-net-9866	0s	128.97 GB	5	526.11 MB (0.4%)	3.0.0-alpha2
✓ klosteroy.uis.nor-net-9866 (10.1.5.215:9866)	klosteroy.uis.nor-net-9866	1s	128.97 GB	5	526.04 MB (0.4%)	3.0.0-alpha2
✓ kongsbakken.uit.nor-net-9866 (10.1.4.193:9866)	kongsbakken.uit.nor-net-9866	2s	128.97 GB	5	526.11 MB (0.4%)	3.0.0-alpha2
✓ lerkendal.ntnu.nor-net-9866 (10.1.9.217:9866)	lerkendal.ntnu.nor-net-9866	2s	128.97 GB	5	526.11 MB (0.4%)	3.0.0-alpha2
✓ mosteroey.uis.nor-net-9866 (10.1.5.197:9866)	mosteroey.uis.nor-net-9866	0s	128.97 GB	5	526.04 MB (0.4%)	3.0.0-alpha2
✓ rennesoey.uis.nor-net-9866 (10.1.5.216:9866)	rennesoey.uis.nor-net-9866	1s	128.97 GB	5	526.04 MB (0.4%)	3.0.0-alpha2
✓ skarven.uit.nor-net-9866 (10.1.4.181:9866)	skarven.uit.nor-net-9866	2s	128.97 GB	5	526.04 MB (0.4%)	3.0.0-alpha2
✓ sokn.uis.nor-net-9866 (10.1.5.194:9866)	sokn.uis.nor-net-9866	0s	128.97 GB	5	526.11 MB (0.4%)	3.0.0-alpha2

Showing 1 to 15 of 15 entries Previous **1** Next

Figure 6.26: Participating nodes - Pyramid,  $n = 15$ ,  $k = 10$ .

```

bakklandet trondheim.uninett.ntnu.nor-net 642
lerkendal bakklandet 2726787
bakklandet klosteroy 807072877
skarven bakklandet 13249269
fjoeloey bakklandet 14504692
byaasen bakklandet 2334005
bakklandet lerkendal 682633015
oesthorn.uninett.simula.nor-net bakklandet 5152
klosteroy bakklandet 14280431
fjoesnisse bakklandet 16796439
kongsbakken bakklandet 8725667
askje bakklandet 14251918
bakklandet bybro 770475562
aunegaarden bakklandet 12642749
bybro bakklandet 5653278
bakklandet kongsbakken 540307474
bakklandet byaasen 537766664
trondheim.uninett.ntnu.nor-net bakklandet 1672
master bakklandet 11110
sokn bakklandet 9648549
bakklandet rennesoey 794550773
bakklandet amundsens 537289403
bakklandet heimdal 766432664
bakklandet aunegaarden 797582618
bakklandet askje 792542198
rennesoey bakklandet 14506598
bakklandet fjoeloey 809610174
mosteroey bakklandet 9911338
bakklandet fjoesnisse 1087379493
bakklandet master 16862
10.1.9.1 bakklandet 836
bakklandet sokn 543545688
heimdal bakklandet 2897860
bakklandet skarven 788020202
bakklandet mosteroey 549374491
bakklandet oesthorn.uninett.simula.nor-net 20320
amundsens bakklandet 8548891
bakklandet 10.1.9.1 321
bakklandet bakklandet 19902

```

Figure 6.27: Traffic logs during encoding (in bytes) - Pyramid,  $n = 15$ ,  $k = 10$ .

```

FSCK started by dr.who (auth:SIMPLE) from /10.1.1.6 for path / at Fri Oct 27 23:24:11 UTC 2017

/PYR_15_10/5GB: Under replicated BP-1039131678-10.1.9.101-1509144996394:blk_-9223372036854775792_1001.
replica(s).

/PYR_15_10/5GB: Under replicated BP-1039131678-10.1.9.101-1509144996394:blk_-9223372036854775776_1002.
replica(s).

/PYR_15_10/5GB: Under replicated BP-1039131678-10.1.9.101-1509144996394:blk_-9223372036854775760_1003.
replica(s).

/PYR_15_10/5GB: Under replicated BP-1039131678-10.1.9.101-1509144996394:blk_-9223372036854775744_1004.
replica(s).

/PYR_15_10/5GB: Under replicated BP-1039131678-10.1.9.101-1509144996394:blk_-9223372036854775728_1005.
replica(s).

Status: HEALTHY
Number of data-nodes: 14
Number of racks: 1
Total dirs: 2
Total symlinks: 0

Replicated Blocks:
Total size: 0 B
Total files: 0
Total blocks (validated): 0
Minimally replicated blocks: 0
Over-replicated blocks: 0
Under-replicated blocks: 0
Mis-replicated blocks: 0
Default replication factor: 1
Average block replication: 0.0
Missing blocks: 0
Corrupt blocks: 0
Missing replicas: 0

Erasure Coded Block Groups:
Total size: 5473128672 B
Total files: 1
Total block groups (validated): 5 (avg. block group size 1094625734 B)
Minimally erasure-coded block groups: 5 (100.0 %)
Over-erasure-coded block groups: 0 (0.0 %)
Under-erasure-coded block groups: 5 (100.0 %)
Unsatisfactory placement block groups: 0 (0.0 %)
Default ecPolicy: RS-DEFAULT-6-3-64k
Average block group size: 14.0
Missing block groups: 0
Corrupt block groups: 0
Missing internal blocks: 5 (6.666665 %)
FSCK ended at Fri Oct 27 23:24:11 UTC 2017 in 46 milliseconds

The filesystem under path '/' is HEALTHY

```

Figure 6.28: Cluster status before reconstruction - Pyramid,  $n = 15$ ,  $k = 10$ .

In a global scenario we can assign each slave node to a stripe index as:

- $c_1 = \text{byaasen (LP1)}$
- $c_2 = \text{bybro (LP1)}$
- $c_3 = \text{heimdal (LP1)}$
- $c_4 = \text{lerkendal (LP1)}$
- $c_5 = \text{askje (LP1)}$
- $c_6 = \text{fjoeloey (LP2)}$
- $c_7 = \text{klosteroeey (LP2)}$
- $c_8 = \text{mosteroey (LP2)}$
- $c_9 = \text{rennesoey (LP2)}$
- $c_{10} = \text{sokn (LP2)}$
- $c_{11} = \text{kongsbakken (LP2)}$
- $c_{12} = \text{amundsen (LP1)}$
- $c_{13} = \text{aunegaarden (GP)}$
- $c_{14} = \text{skarven (GP)}$
- $\langle \text{Missing node} \rangle \text{ (GP)}$
- $c_{15} = \text{bjoervika (Recovery)}$

```

bakklandet trondheim.uninett.ntnu.nornet 1605
bjoervika askje 3858347
lerkendal bakklandet 59809
bjoervika aunegaarden 2040230
bakklandet klosteroeey 22904
skarven bakklandet 74553
fjoeloey bakklandet 73924
byaasen bakklandet 47404
bakklandet lerkendal 21731
bjoervika skarven 3531605
oesthorn.uninett.simula.nornet bakklandet 7733
klosteroeey bakklandet 73298
bjoervika rennesoey 2933711
kongsbakken bakklandet 77487
kongsbakken bjoervika 335569447
mosteroey bjoervika 358756549
skarven bjoervika 787677400
askje bakklandet 61716
bakklandet bybro 17721
aunegaarden bakklandet 76596
bybro bakklandet 42199
bybro bjoervika 65727987
bakklandet kongsbakken 18594
rennesoey bjoervika 356386686
bakklandet byaasen 18652
trondheim.uninett.ntnu.nornet bakklandet 4180
bjoervika fjoeloey 2885131
bakklandet rennesoey 22195
sokn bakklandet 71249
amundsen bjoervika 732068180
bjoervika amundsen 3857041
lerkendal bjoervika 794565659
bakklandet amundsen 18666
bakklandet heimdal 19573
bakklandet aunegaarden 18638
fjoeloey bjoervika 358465301
bjoervika sokn 4774183
bakklandet askje 21955
rennesoey bakklandet 74147
bjoervika mosteroey 2121539
bakklandet fjoeloey 23040
mosteroey bakklandet 73892
aunegaarden bjoervika 386796479
klosteroeey bjoervika 360469565
byaasen bjoervika 421133007
bjoervika klosteroeey 2812183
heimdal bjoervika 656284229
sokn bjoervika 684567566
bakklandet sokn 20187
bjoervika bybro 6362922
heimdal bakklandet 58725
askje bjoervika 513160121
bakklandet skarven 18882
bakklandet mosteroey 21904
bakklandet oesthorn.uninett.simula.nornet 30380
bjoervika lerkendal 8240729
bjoervika byaasen 3573898
amundsen bakklandet 77278
bjoervika kongsbakken 1777462
bjoervika heimdal 6088386
bjoervika bjoervika 360347572

```

Figure 6.29: Traffic logs during reconstruction (in bytes) - Pyramid,  $n = 15$ ,  $k = 10$ .

A global recovery process will require transmissions from  $k = 10$  nodes in order to be completed successfully. After normalizing our data to  $\frac{14}{10}$ , we can obtain an average of all the traffic needed to fulfill this process as:

$$\frac{\sum_{\forall G \subseteq \{1, \dots, n-1\}: |G|=k} \sum_{i \in G} (t_i^{\text{in}} + t_i^{\text{out}})}{\binom{n-1}{k}} = 8011\,158\,379 \text{ bytes (7.46 GB)}.$$

If we instead needed to do a recovery process on the first local parity group, we could then calculate the average necessary traffic as follows:

$$\frac{\sum_{g \in G_1 = \{1, 2, 3, 4, 5, 12\}} \sum_{i \in G_1 \setminus \{g\}} (t_i^{\text{in}} + t_i^{\text{out}})}{|G_1|} = 4\,801\,772\,830 \text{ bytes (4.47 GB)}.$$

For the second local parity group, we can similarly obtain it as:

$$\frac{\sum_{g \in G_2 = \{6, 7, 8, 9, 10, 11\}} \sum_{i \in G_2 \setminus \{g\}} (t_i^{\text{in}} + t_i^{\text{out}})}{|G_2|} = 3\,244\,455\,073 \text{ bytes (3.02 GB)}.$$

Knowing that in our  $n = 15, k = 10$  code there are two local groups, each of size 6, as well as three global parities, we can align these three results in a general case as:

$$\frac{6 \times 4\,801\,772\,830 + 6 \times 3\,244\,455\,073 + 3 \times 8\,011\,158\,379}{15} = 4\,820\,722\,837 \text{ bytes (4.49 GB)}.$$

From our results we can state that, in average, 4.49 GB of transferred data is required in order to reconstruct any missing node.

```

FSCK started by dr.who (auth:SIMPLE) from /10.1.1.6 for path / at Fri Oct 27 23:10:46 UTC 2017

Status: HEALTHY
Number of data-nodes: 15
Number of racks:      1
Total dirs:           2
Total symlinks:       0

Replicated Blocks:
Total size: 0 B
Total files: 0
Total blocks (validated): 0
Minimally replicated blocks: 0
Over-replicated blocks: 0
Under-replicated blocks: 0
Mis-replicated blocks: 0
Default replication factor: 1
Average block replication: 0.0
Missing blocks: 0
Corrupt blocks: 0
Missing replicas: 0

Erasure Coded Block Groups:
Total size: 5473128672 B
Total files: 1
Total block groups (validated): 5 (avg. block group size 1094625734 B)
Minimally erasure-coded block groups: 5 (100.0 %)
Over-erasure-coded block groups: 0 (0.0 %)
Under-erasure-coded block groups: 0 (0.0 %)
Unsatisfactory placement block groups: 0 (0.0 %)
Default ecPolicy: RS-DEFAULT-6-3-64k
Average block group size: 15.0
Missing block groups: 0
Corrupt block groups: 0
Missing internal blocks: 0 (0.0 %)
FSCK ended at Fri Oct 27 23:10:46 UTC 2017 in 6 milliseconds

The filesystem under path '/' is HEALTHY

```

Figure 6.30: Cluster status after reconstruction - Pyramid,  $n = 15, k = 10$ .

## 6.3 Results Summary

This section provides an overall summary of the results we have encountered. The main variables that are included in this comparison are repair bandwidth and duration. Note that Tables 6.2 to 6.5 must be read from left to right, proportioning policies on each row to those located above in each column.



Policy	Process	Traffic (bytes)	Traffic (GB)	Duration (sec.)
RS(5, 3)	Encoding	10 750 953 930	10.01	700
RS(5, 3)	Reconstruction	9 754 113 842	9.08	546
RS(11, 8)	Encoding	7 494 812 653	6.98	506
RS(11, 8)	Reconstruction	8 197 198 761	7.63	370
RS(14, 10)	Encoding	7 644 837 886	7.12	554
RS(14, 10)	Reconstruction	5 263 847 902	4.90	168
PYR(6, 3)	Encoding	13 616 668 524	12.68	633
PYR(6, 3)	Reconstruction	7 911 332 440	7.37	572
PYR(12, 8)	Encoding	9 930 213 333	9.25	503
PYR(12, 8)	Reconstruction	5 493 983 409	5.12	306
PYR(15, 10)	Encoding	10 955 338 584	10.20	558
PYR(15, 10)	Reconstruction	4 820 722 837	4.49	271

Table 6.1: Results summary - Network traffic and duration.

Policy	RS(5, 3)	RS(11, 8)	RS(14, 10)	PYR(6, 3)	PYR(12, 8)	PYR(15, 10)
RS(5, 3)	-	143.4%	140.6%	78.9%	108.2%	98.1%
RS(11, 8)	69.7%	-	98.0%	55.0%	75.4%	68.4%
RS(14, 10)	71.1%	102.0%	-	56.1%	77.0%	69.7%
PYR(6, 3)	126.6%	181.6%	178.1%	-	137.1%	124.2%
PYR(12, 8)	92.3%	132.4%	129.8%	72.9%	-	90.6%
PYR(15, 10)	101.9%	146.1%	143.3%	80.4%	110.3%	-

Table 6.2: Network traffic comparison during encoding.

Policy	RS(5, 3)	RS(11, 8)	RS(14, 10)	PYR(6, 3)	PYR(12, 8)	PYR(15, 10)
RS(5, 3)	-	119.0%	185.3%	123.3%	177.5%	202.3%
RS(11, 8)	84.0%	-	155.7%	103.6%	149.2%	170.0%
RS(14, 10)	54.0%	64.2%	-	66.5%	95.8%	109.2%
PYR(6, 3)	81.1%	96.5%	150.3%	-	144.0%	164.1%
PYR(12, 8)	56.3%	67.0%	104.4%	69.4%	-	114.0%
PYR(15, 10)	49.4%	58.8%	91.6%	60.9%	87.7%	-

Table 6.3: Network traffic comparison during reconstruction.

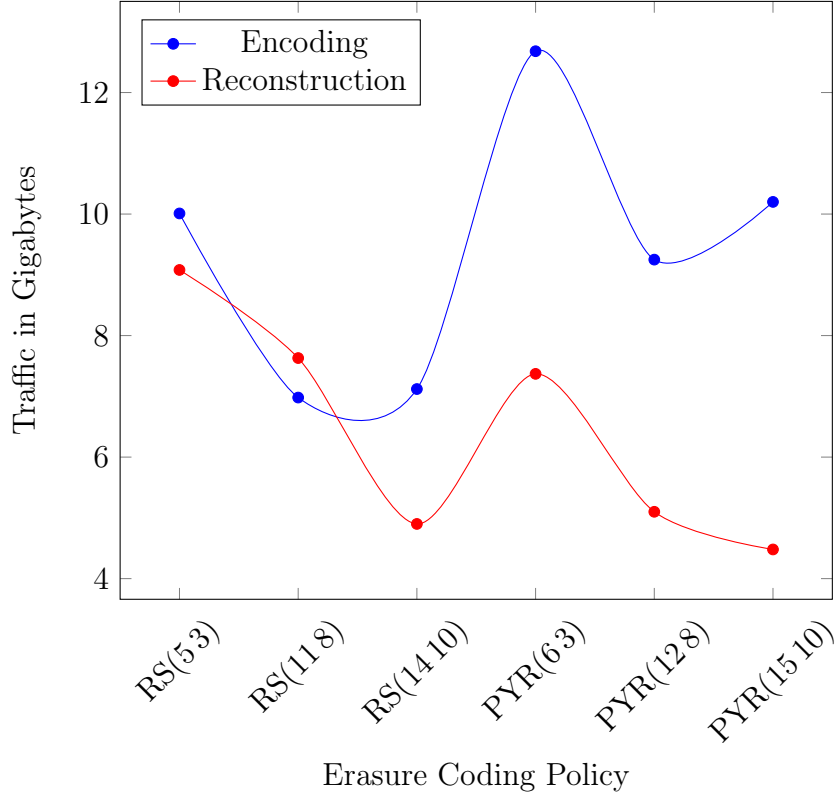


Figure 6.31: Network traffic comparison.

Policy	RS(5, 3)	RS(11, 8)	RS(14, 10)	PYR(6, 3)	PYR(12, 8)	PYR(15, 10)
RS(5, 3)	-	138.3%	126.4%	110.6%	139.2%	125.4%
RS(11, 8)	72.3%	-	91.3%	79.9%	100.6%	90.7%
RS(14, 10)	79.1%	109.5%	-	87.5%	110.1%	99.3%
PYR(6, 3)	90.4%	125.1%	114.3%	-	125.8%	113.4%
PYR(12, 8)	71.9%	99.4%	90.8%	79.5%	-	90.1%
PYR(15, 10)	79.7%	110.3%	100.7%	88.2%	110.9%	-

Table 6.4: Duration comparison during encoding.

Policy	RS(5, 3)	RS(11, 8)	RS(14, 10)	PYR(6, 3)	PYR(12, 8)	PYR(15, 10)
RS(5, 3)	-	147.6%	325.0%	95.5%	178.4%	201.5%
RS(11, 8)	67.8%	-	220.2%	64.7%	120.9%	136.5%
RS(14, 10)	30.8%	45.4%	-	29.4%	54.9%	62.0%
PYR(6, 3)	104.8%	154.6%	340.5%	-	186.9%	211.1%
PYR(12, 8)	56.0%	82.7%	182.1%	53.5%	-	112.9%
PYR(15, 10)	38.7%	73.2%	161.3%	47.4%	88.6%	-

Table 6.5: Duration comparison during reconstruction.

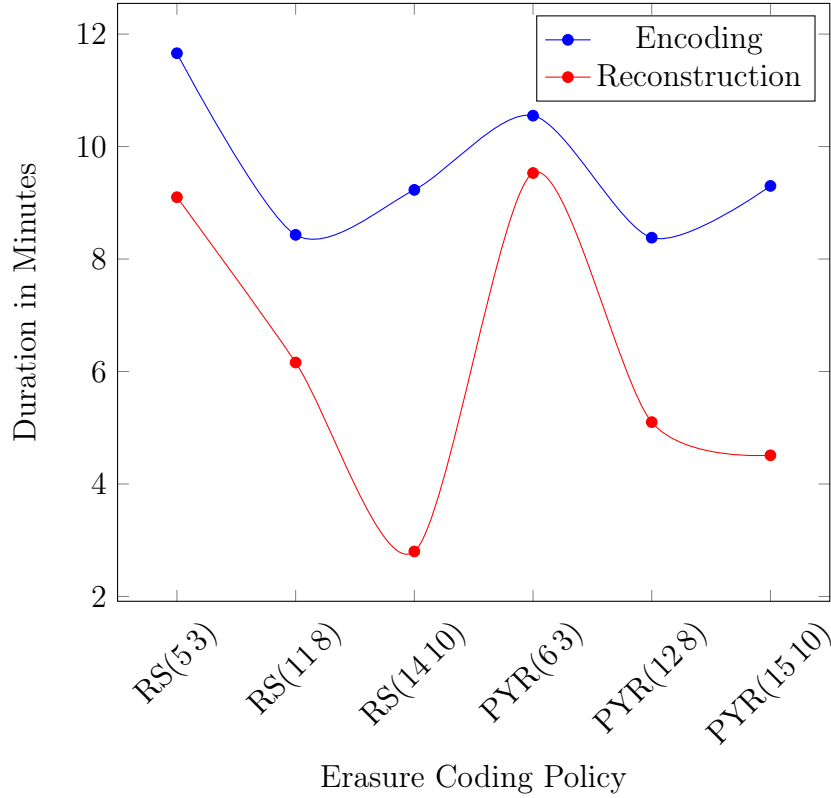


Figure 6.32: Duration comparison.

Table 6.6 describes the ability of each code to do several simultaneous repairs, following the scheme below:

- Global: suggests that the code is able to retrieve the missing blocks through a global structure, using at least one global parity.
- Local: points out that the code is capable of recovering the code through its local parities, reducing the network traffic significantly.
- Local/Global: missing blocks are reconstructed either locally or globally, depending on its stripe position.
- Restricted: reconstruction is possible only for very specific scenarios.
- No: if none of these apply, the code is not able to recover the missing blocks.

Repairs	1	2	3	4	5
RS(5, 3)	Global	Global	No	No	No
RS(11, 8)	Global	Global	Global	No	No
RS(14, 10)	Global	Global	Global	Global	No
PYR(6, 3)	Local/Global	Local/Global	Restricted	No	No
PYR(12, 8)	Local/Global	Local/Global	Global	Restricted	No
PYR(15, 10)	Local/Global	Local/Global	Global	Global	Restricted

Table 6.6: Repair comparison.



# Chapter 7

## Conclusion and Further Work

This thesis confirms that Apache Hadoop allows policy alteration to some extent, and that this open-source software is totally compatible with NorNet Core. It has been an excellent scenario to obtain pragmatic results in a real distributed scheme, including nodes within miles from one another, without considerable inconveniences.

According to our results we can conclude that in all cases, bigger codes always offer higher efficiency through network traffic, recovery span and fault-tolerance, and they should be preferred over smaller codes as long as there exists no constraining factors (such as cluster size). As expected, Pyramid codes helped to reduce network traffic sent and received during reconstruction processes, although yielding no improvement in its duration, since intervening nodes need to individually send the same amount of information as in RS policies. An interesting result found in this thesis is that, disregarding generated local parities, policy  $\text{PYR}(6, 3)$  still offered a higher repair bandwidth than larger RS codes such as  $\text{RS}(14, 10)$ .

Suggested further work could be experimenting with new erasure coding policies and comparing properties other than duration and network traffic, finding ways to more accurately control recovery processes within Hadoop, analyzing interactions between more distant sites to see how less homogeneous clusters affect performance, using vector codes as opposed to scalar codes, and investigating codes that yield a higher security level.



# Chapter 8

## Appendix - Code Snippets

### 8.1 Log-Harvesting Script

```
#!/bin/bash

touch output

startTime=${1:-null}
endTime=${2:-null}

echo $startTime
echo $endTime

if [ -f finalOutput ]; then
    rm finalOutput
fi

if [ -f partialOutput ]; then
    rm partialOutput
fi

touch partialOutput
touch finalOutput

for i in $(cat nodes); do
    echo $i
    scp srl_sards@$i:/var/log/iptraf-ng/hadoop.log $i.log
done

for i in $(cat nodes); do
    while read line; do
        lineTime=$(echo $line | awk '{print $4}')
        if [[ $(echo $line | awk '{print $15}') == "sent;" ]]; then
            flagCountA=0
            flagCountB=0
            if [ $startTime == null ] || [ $startTime \< $lineTime ]; then
                if [ $endTime == null ] || [ $endTime \> $lineTime ]; then
                    fromLine=$(echo $line | awk '{print $11}')
                    if grep -Fxq "$(echo $fromLine | awk -F'.' '{print $2}')"
                        node_names; then
                        from=$(echo $fromLine | awk -F'.' '{print $2}')
                        flagCountA=1
                    fi
                fi
            fi
        fi
    done
done
```

```

fi
if grep -Fxq "$(echo $fromLine | awk -F'.' '{print $1}')"
    node_names; then
    from=$(echo $fromLine | awk -F'.' '{print $1}')
    flagCountA=1
fi
if [ $flagCountA -eq 0 ]; then
    from=$(echo $fromLine | awk -F':' '{print $1}')
fi
toLine=$(echo $line | awk '{print $13}')
if grep -Fxq "$(echo $toLine | awk -F'.' '{print $2}')"
    node_names; then
    to=$(echo $toLine | awk -F'.' '{print $2}')
    flagCountB=1
fi
if grep -Fxq "$(echo $toLine | awk -F'.' '{print $1}')"
    node_names; then
    to=$(echo $toLine | awk -F'.' '{print $1}')
    flagCountB=1
fi
if [ $flagCountB -eq 0 ]; then
    to=$(echo $toLine | awk -F':' '{print $1}')
fi
bytes=$(echo $line | awk '{print $18}')
if [ $flagCountA -eq 1 ] && [ $flagCountB -eq 1 ]; then
    bytes=$((bytes/2))
fi
echo $from $to $bytes >> partialOutput
fi
fi
done < $i.log
done

awk -F' ' '{array[$1" "$2]+=$3} END { for (i in array)
    {print i" " array[i]}}' partialOutput > finalOutput

```

## 8.2 Hadoop's Default Cauchy Matrix Generator

```

/**
 * Ported from Intel ISA-L library.
 */
public static void genCauchyMatrix(byte[] a, int m, int k) {
    // Identity matrix in high position
    for (int i = 0; i < k; i++) {
        a[k * i + i] = 1;
    }

    // For the rest choose 1/(i + j) | i != j
    int pos = k * k;
    for (int i = k; i < m; i++) {
        for (int j = 0; j < k; j++) {
            a[pos++] = GF256.gfInv((byte) (i ^ j));
        }
    }
}
}

```



## 8.3 Hadoop's Modified Cauchy Matrix Generator

```
public static void genCauchyMatrix(byte[] a, int m, int k) {
    // Identity matrix in high position
    for (int i = 0; i < k; i++) {
        a[k * i + i] = 1;
    }

    // For the rest choose 1/(i + j) | i != j
    int pos = k * k;
    for (int i = k; i < m; i++) {
        for (int j = 0; j < k; j++) {
            if ((i-k) < ((m-k)/2)) {
                if (pos%2 == 0) {
                    a[pos++] = 0;
                }
                else {
                    a[pos++] = GF256.gfInv((byte) (i ^ j));
                }
            }
            else {
                if (pos%2 == 1) {
                    a[pos++] = 0;
                }
                else {
                    a[pos++] = GF256.gfInv((byte) (i ^ j));
                }
            }
        }
        System.out.println(a[pos-1]);
    }
}
```

## 8.4 Hadoop's Modified Pyramid Policy

```
@Override
protected void doDecode(ByteArrayDecodingState decodingState) {
    byte[] output = decodingState.outputs[0];
    int dataLen = decodingState.decodeLength;
    CoderUtil.resetOutputBuffers(decodingState.outputs,
        decodingState.outputOffsets, dataLen);

    int erasedIdx = decodingState.erasedIndexes[0];

    int localDataUnits = 0;
    int localAllUnits = 0;

    if (getNumDataUnits()%2 == 1) {
        localDataUnits = getNumDataUnits() + 1;
    }
    else {
        localDataUnits = getNumDataUnits();
    }
    if (getNumAllUnits()%2 == 1) {
        localAllUnits = getNumAllUnits() + 1;
    }
}
```

```

else {
    localAllUnits = getNumAllUnits();
}

int[] targetIdxArray = new int[localAllUnits/2];

if ((erasedIdx < localDataUnits/2) ||
    (erasedIdx == (getNumDataUnits() + 1))) {
    for (int i=0;i < (localDataUnits/2);i++){
        targetIdxArray[i]=i;
    }
    targetIdxArray[targetIdxArray.length - 1] = getNumDataUnits() + 1;
}
else if (erasedIdx < (getNumDataUnits()+1)) {
    int j=0;
    for (int i=(localDataUnits/2);i < ( getNumDataUnits() + 1);i++){
        targetIdxArray[j]=i;
        j++;
    }
}

if (erasedIdx > (getNumDataUnits()+1)) {
    byte[][] realInputs = new byte[getNumDataUnits()][];
    int[] realInputOffsets = new int[getNumDataUnits()];
    for (int i = 0; i < getNumDataUnits(); i++) {
        realInputs[i] =
            decodingState.inputs[validIndexes[i]];
        realInputOffsets[i] =
            decodingState.inputOffsets[validIndexes[i]];
    }
    RSUtil.encodeData(gfTables, dataLen, realInputs, realInputOffsets,
        decodingState.outputs, decodingState.outputOffsets);
}
else {
    int[] coefficientsArray = new int[getNumAllUnits()];
    int j=0;
    for (int i=getNumDataUnits()*getNumDataUnits(); i <
        encodeMatrix.length; i++){
        if (i%(getNumAllUnits()-getNumDataUnits()) == 0 ||
            i%(getNumAllUnits()-getNumDataUnits()) == 1) {
            if (encodeMatrix[i] != 0) {
                coefficientsArray[j] = encodeMatrix[i];
                j++;
            }
        }
    }
    for (int i=j;i<coefficientsArray.length;i++) {
        coefficientsArray[i] = 1;
    }
    System.out.println("Coefficients Array... ");
    for (int i=0;i<coefficientsArray.length;i++){
        System.out.println("Coefficient: " + coefficientsArray[i]);
    }
    int targetIdx;
    for (int eIdx = 0; eIdx < targetIdxArray.length; eIdx++){
        targetIdx = targetIdxArray[eIdx];
        if ((targetIdx == erasedIdx) || (eIdx != 0 && targetIdx == 0))
            continue;
}

```

```

int iIdx, oIdx;
for (iIdx = decodingState.inputOffsets[targetIdx],
     oIdx = decodingState.outputOffsets[0];
     iIdx < decodingState.inputOffsets[targetIdx] + dataLen;
     iIdx++, oIdx++) {
    output[oIdx] ^= GF256.gfMul(decodingState.inputs[targetIdx][iIdx],
                               (byte) coefficientsArray[targetIdx]);
    output[oIdx] = GF256.gfMul(output[oIdx],
                               GF256.gfInv((byte) coefficientsArray[erasedIdx]));
}
}
}
}

```

# Bibliography

- [1] Savvycom Software, “What you need to know about Hadoop and its ecosystem.” <https://savvycomsoftware.com/what-you-need-to-know-about-hadoop-and-its-ecosystem>, January 2016.
- [2] P. Vagata and K. Wilfong, “Scaling the Facebook data warehouse to 300 PB,” tech. rep., Facebook, April 2014.
- [3] Cisco, “The zettabyte era: Trends and analysis,” tech. rep., June 2017.
- [4] F. Schmuck and R. Haskin, “GPFS: A shared-disk file system for large computing clusters,” in *Proceedings of the 1st USENIX Conference on File and Storage Technologies (FAST)*, January 2002.
- [5] H. Weatherspoon and J. Kubiatowicz, “Erasure coding vs. replication: A quantitative comparison,” *Revised Papers from the First International Workshop on Peer-to-Peer Systems (IPTPS)*, pp. 328–338, March 2002.
- [6] J. Cook, R. Primmer, and A. de Kwant, “Comparing cost and performance of replication and erasure coding.” arXiv:1308.1887, August 2013.
- [7] K. M. Greenan, E. L. Miller, and J. J. Wylie, “Reliability of flat XOR-based erasure codes on heterogeneous devices,” in *Proceedings of the IEEE International Conference on Dependable Systems and Networks With FTCS and DCC*, (Anchorage, AK, USA), June 2008.
- [8] K. M. Greenan, X. Li, and J. J. Wylie, “Flat XOR-based erasure codes in storage systems: Constructions, efficient recovery, and tradeoffs,” in *Proceedings of the 26th IEEE Symposium on Mass Storage Systems and Technologies (MSST)*, pp. 1–14, May 2010.
- [9] K. V. Rashmi, N. B. Shah, D. Gu, H. Kuang, D. Borthakur, and K. Ramchandran, “A “hitchhiker’s” guide to fast and efficient data reconstruction in erasure-coded data centers,” in *Proceedings of the ACM Conference on SIGCOMM*, pp. 331–342, August 2014.
- [10] M. Sathiamoorthy, M. Asteris, D. Papailiopoulos, A. G. Dimakis, R. Vadali, S. Chen, and D. Borthakur, “XORing elephants: Novel erasure codes for big data,” *Proceedings of the VLDB Endowment*, vol. 6, pp. 325–336, March 2013.
- [11] A. G. Dimakis, P. B. Godfrey, Y. Wu, M. Wainwright, and K. Ramchandran, “Network coding for distributed storage systems,” *IEEE Transactions on Information Theory*, vol. 56, pp. 4539–4551, August 2010. arXiv:0803.0632.

- [12] A. G. Dimakis, K. Ramchandran, Y. Wu, and C. Suh, “A survey on network codes for distributed storage,” *Proceedings of the IEEE*, vol. 99, pp. 476–489, March 2011.
- [13] M. Ye and A. Barg, “Repairing Reed-Solomon codes: Universally achieving the cut-set bound for any number of erasures.” arXiv:1706.00112v1, May 2017.
- [14] N. B. Shah, K. V. Rashmi, P. V. Kumar, and K. Ramchandran, “Distributed storage codes with repair-by-transfer and nonachievability of interior points on the storage-bandwidth tradeoff,” *IEEE Transactions on Information Theory*, vol. 58, pp. 1837–1852, March 2012.
- [15] T. Ernvall, “Codes between MBR and MSR points with exact repair property,” *IEEE Transactions on Information Theory*, vol. 60, pp. 6993–7005, August 2014. arXiv:1312.5106.
- [16] K. V. Rashmi, N. B. Shah, and P. V. Kumar, “Optimal exact-regenerating codes for distributed storage at the MSR and MBR points via a product-matrix construction,” *IEEE Transactions on Information Theory*, vol. 57, pp. 5227–5239, July 2011. arXiv:1005.4178.
- [17] S. Goparaju, A. Fazeli, and A. Vardy, “Minimum storage regenerating codes for all parameters,” *IEEE Transactions on Information Theory*, vol. 63, pp. 6318–6328, October 2017.
- [18] V. Guruswami and A. S. Rawat, “MDS code constructions with small sub-packetization and near-optimal repair bandwidth,” in *Proceedings of the 28th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pp. 2109–2122, January 2017.
- [19] A. S. Rawat, I. Tamo, V. Guruswami, and K. Efremenko, “MDS code constructions with small sub-packetization and near-optimal repair bandwidth.” arxiv:1709.08216v1, September 2017.
- [20] D. S. Papailiopoulos and A. G. Dimakis, “Locally repairable codes,” in *Proceedings of the IEEE International Symposium on Information Theory (ISIT)*, (Cambridge, MA, USA), pp. 2771–2775, July 2012. arXiv:1206.3804.
- [21] P. Gopalan, C. Huang, H. Simitci, and S. Yekhanin, “On the locality of codeword symbols,” *IEEE Transactions on Information Theory*, vol. 58, pp. 6925–6943, November 2012.
- [22] A. S. Rawat, O. O. Koyluoglu, N. Silberstein, and S. Vishwanath, “Optimal locally repairable and secure codes for distributed storage systems,” *IEEE Transactions on Information Theory*, vol. 60, pp. 212–236, January 2014. arXiv:1210.6954.
- [23] I. Tamo and A. Barg, “A family of optimal locally recoverable codes,” *IEEE Transactions on Information Theory*, vol. 60, pp. 4661–4676, August 2014.
- [24] Y. Zhou, “Ceph erasure coding introduction.” <https://software.intel.com/en-us/blogs/2015/04/06/ceph-erasure-coding-introduction>, April 2015.

- [25] C. Huang, H. Simitci, Y. Xu, A. Ogus, B. Calder, P. Gopalan, J. Li, and S. Yekhanin, “Erasure coding in Windows Azure storage,” in *Proceedings of the USENIX Annual Technical Conference*, (Boston, MA, USA), June 2012.
- [26] A. Datta and F. Oggier, “An overview of codes tailor-made for better repairability in networked distributed storage systems,” *ACM SIGACT News*, vol. 44, pp. 89–105, March 2013. arXiv:1109.2317.
- [27] M. Xia, M. Saxena, M. Blaum, and D. A. Pease, “A tale of two erasure codes in HDFS,” in *Proceedings of the 13th USENIX Conference on File and Storage Technologies (FAST)*, pp. 213–226, February 2015.
- [28] C. Huang, M. Chen, and J. Li, “Pyramid codes: Flexible schemes to trade space for access efficiency in reliable data storage systems,” *ACM Transactions on Storage*, vol. 9, March 2013.
- [29] E. G. Gran, T. Dreibholz, and A. Kvalbein, “NorNet Core - A multi-homed research testbed,” *Computer Networks: The International Journal of Computer and Telecommunications Networking*, vol. 61, pp. 75–87, March 2014.
- [30] T. Dreibholz, “The NorNet Core handbook.” <https://home.simula.no/~dreibh/NorNet-Core-Handbook.pdf>, June 2016.