DYNAMIC ERASURE CODING POLICY ALLOCATION AND DYNAMIC DATA REPLICATION IN HADOOP 3.0

IT7611- CREATIVE AND INNOVATIVE PROJECT

A PROJECT REPORT

Submitted by

S.ThirumagalDhivya 2017506613

C.Ramkumar 2017506581

in partial fulfillment for the award of the degree

of

BACHELOR OF TECHNOLOGY

IN

INFORMATION TECHNOLOGY

DEPARTMENT OF INFORMATION TECHNOLOGY

ANNA UNIVERSITY

MADRAS INSTITUTE OF TECHNOLOGY CAMPUS, CHENNAI-600 044

APRIL-2020

ANNA UNIVERSITY:: CHENNAI 600 044

BONAFIDE CERTIFICATE

Certified that this project report **"DYNAMIC ERASURE CODING POLICY ALLOCATION AND DYNAMIC DATA REPLICATION IN HADOOP 3.0"** is the Bonafide work of **THIRUMAGALDHIVYA S (2017506613)** , **RAMKUMAR C (2017506581)** who carried out the project work under my supervision. Certified further that to the best of my knowledge the work reported here does not form part of any other thesis or dissertation on the basis of which a degree or award was conferred on an earlier occasion on this or any other candidate.

SIGNATURE	**SIGNATURE**
DR.DHANANJAY KUMAR,	**DR.B.LYDIA ELIZABETH,**
HEAD OF THE DEPARTMENT,	**SUPERVISOR,**
	PROFESSOR,
DEPT. OF INFORMATION TECHNOLOGY,	DEPT. OF INFORMATION TECHNOLOGY,
MADRAS INSTITUTE OF TECHNOLOGY, ANNA UNIVERSITY,CHROMPET,	MADRAS INSTITUTE OF TECHNOLOGY, ANNA UNIVERSITY,CHROMPET,
CHENNAI-600 044.	CHENNAI-600 044.

ACKNOWLEDGEMENT

The satisfaction that accompanies the successful completion of our task would be incomplete without mentioning the names of people who made it possible, whose constant guidance and encouragement crowns all efforts with success .We express our gratitude and sincere thanks to our respected Dean ,**DR.THYAGARAJAN.T** , Anna University, MIT Campus for providing the excellent computing facilities to do the project . Our heartfelt thanks to **DR.DHANANJAY KUMAR**, Head, Department of Information Technology, Madras Institute of Technology Campus, Anna University, for the prompt and limitless help in providing the excellent computing facilities to do the project.

We express our profound sense of gratitude to our project panel member **DR.B.LYDIA ELIZABETH** Department of Information Technology for their invaluable support, guidance and encouragement for the successful completion of this project.

ABSTRACT

Erasure Code (EC) is being tipped to be the next best alternative to Replication for providing redundancy for Cloud Storage. Already major players like Microsoft and Facebook are having initial implementations using Erasure Code. Hadoop 0.20 was the first version that supported Erasure Code (aka HDFS-RAID), but EC was not included in later versions, only to resurface in Hadoop 3.0 (HDFS-EC). Hadoop 3.0.0 supports three default Erasure Code policies which are RS(3,2), RS(6,3) and RS(10,4). To have greater flexibility, in this work we opt for the implementation of new Erasure Code policies [RS(4,3), RS(5,3), RS(7,3), RS(7,4), RS(8,4), RS(9,4)] and the development of a Dynamic Erasure Coding Policy Allocation, based on minimum overhead produced, in order to maximize storage capacity. Three types of dynamic allocation have been proposed and implemented. A performance evaluation of the new policies was conducted in order to find the optional one for a NAS/SAN architecture and the effectiveness of the three implemented Dynamic Allocation of EC policy is provided.

HDFS ensures availability of data by replicating data to different nodes. However, the replication policy of HDFS does not consider the popularity of data. The popularity of the files tend to change over time. Hence, maintaining a fixed replication factor will affect the storage efficiency of HDFS. In this project, we propose an efficient dynamic data replication management system, which considers the popularity of files stored in HDFS before replication. This strategy dynamically classifies the files to hot data or cold data based on its popularity and increases the replica of hot data by applying erasure coding for cold data. The experiment results show that the proposed method effectively reduces the storage utilization up to 40% without affecting the availability and fault tolerance in HDFS.

TABLE OF CONTENTS

CHAPTER-1

INTRODUCTION

1.1OVERVIEW

Hadoop [1][2] (aka Apache Hadoop) is an open-source software designed to operate on networked computers (cluster) to manipulate massive amounts of data. The core of Hadoop is firstly its file system known as Hadoop File System (HDFS) and secondly, its programming framework known as Map-Reduce programming model. Hadoop was first released in 2011 and its latest stable release is Hadoop 3.0.0, launched in December 2017. During that period of time, a lot of enhancements have seen applied to Hadoop. Examples of enhancement are: Yarn, Oozie, Hive, Pig-Latin. Hadoop has since been widely adopted, by major cloud providers such as Yahoo and Facebook, just to mention a few.

Big Data is high-volume, high-velocity and high-variety information that demands cost-effective, innovative forms of information processing for enhanced insight and decision making [1]. The extraction, storage and processing of big data is beyond the ability of traditional data processing techniques. Therefore a more sophisticated framework is required to handle these data. Apache Hadoop [2] is one of the best known platforms for distributed storing and processing of big data across clusters of computers. The storage component of Hadoop, Hadoop Distributed File System (HDFS)[3][4] maintains a default replication factor for each file as three, which is placed in separate nodes.

<u>1.2 PROBLEM STATEMENT</u>

Hadoop's defacto Redundancy scheme is the expensive Replication – the default 3x replication scheme in HDFS has 200% overhead in storage space and other resources such as CPU consumption and network bandwidth. Despite that the replicas are used only in cases of failures. However they need to be available as and when needed, therefore need to be live.

<u>1.3 EXISTING SYSTEM</u>

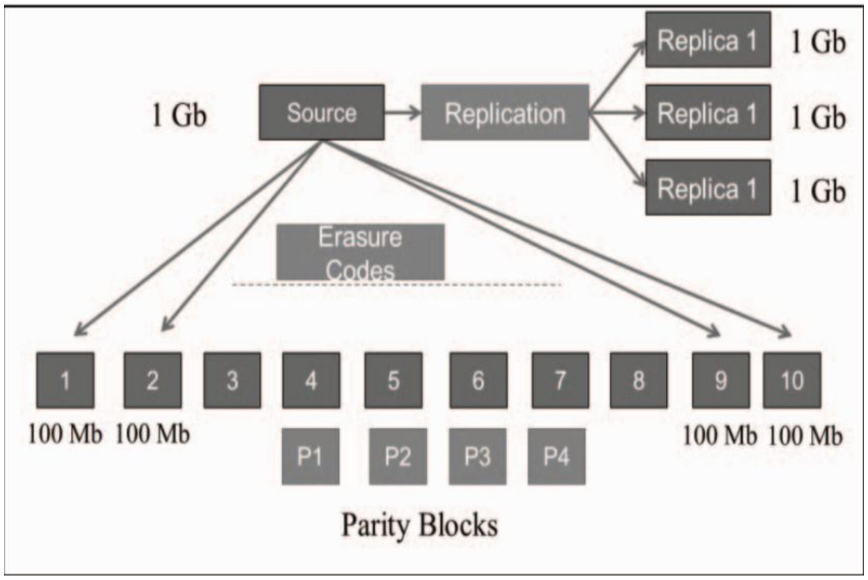
HDFS provides high performance access to data by applying a static and default replication strategy. Though HDFS ensures high reliability, scalability and high availability, its static and default approach in data replication requires large amounts of storage space. With a replication factor of three, a file is copied three times in different nodes. If the size of a file is 1TB then, after replication it will take 3TB of space. Furthermore, in HDFS the files are replicated without considering the popularity of the file. In real scenario, the access frequency of every file in the file system is not accessed equally. Some files are accessed frequently while some others stay idle for a long period of time. By keeping replicas for these idle files, a valuable amount of storage space will be consumed unnecessarily resulting in wastage of storage space, that lead to bad effect of performance. If the number of copies can be reduced for these files the storage space can be freed and can be utilized by more frequently accessed files. But reducing the number of copies increases the chance for data loss. Therefore a data replication strategy which reduces the replicas of under-utilized file without affecting the data availability and fault tolerance has to be implemented.

DISADVANTAGE OF EXISTING SYSTEM:

- High storage overhead
- Low data durability

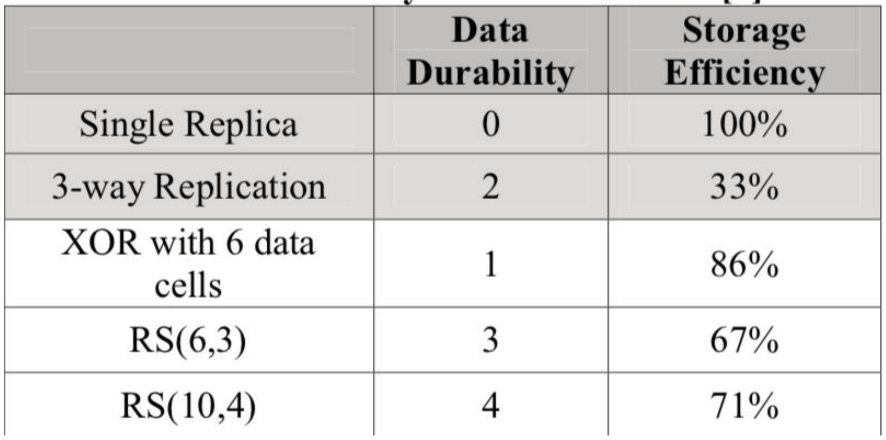
1.4 PROPOSED SYSTEM:

An alternative solution is to use Erasure Coding (EC) in lieu of replication, as EC provides the same level of fault- tolerance with much less storage space. In typical Erasure Coding (EC) setups, the storage overhead is no more than 50%.Erasure Coding can be divided into 3 sub-processes: Splitting, Encoding and Placing. Any file is split into n blocks, and then encoded to produce m parity blocks. All the blocks are placed in different datanode, so as to minimize block lost in cases of HDD failures.



The effectiveness of Erasure Code compared to Replication is summarized in the table below:

Table 1. Efficiency of Erasure Codes [3]



	Data Durability	Storage Efficiency
Single Replica	0	100%
3-way Replication	2	33%
XOR with 6 data cells	1	86%
RS(6,3)	3	67%
RS(10,4)	4	71%

In the proposed work, we also present a dynamic data replication strategy which focuses on a storage efficient replication in HDFS without affecting the availability of data. In this strategy data files are classified into hot and cold based on the popularity of the data file in the Hadoop cluster. The replica of the popular file is increased while the replica of non-popular file is reduced to one and erasure coding is applied on it to prevent data loss. The result shows that the proposed replication strategy reduces the storage space utilization significantly without affecting the availability constraint of HDFS.

<u>ADVANTAGES OF PROPOSED SYSTEM:</u>

- Dynamically allocating erasure coding policies
- Dynamically allocating replication factor based on popularity index.
- Low storage overhead
- High data durability

CHAPTER-2

LITERATURE SURVEY

Apart from supporting Erasure Coding, Hadoop 3.0 has adopted a series of changes as follows: usage of Java 8, support for opportunistic containers, shell script rewrite, support for more than 2 namenodes, and support for file system connector and intra-datanode balancer [13].

Incorporating Erasure Code again in Hadoop makes sense as data can be read/write in a pipeline from different data nodes simultaneously. Working in a pipeline, the blocks can be striped and distributed fairly amongst the different datanodes following the rack-aware [14] principle.

The first major change in Hadoop 3.0 is the architecture for Erasure Code. To reduce memory consumption of Namenode keeping track of the locations of each block, a new block naming protocol is used. HDFS-RAID was using a block ID which is allocated sequentially depending on the time of creation of the block. HDFS-EC also uses Block ID, which consists of a flag (0 = contiguous/1 = striped), a middle section, which indicates the logical block and a last part which represents the index of the storage block in the logical block. In doing so, a namenode is able to manage a logical block (Data + Parity Blocks) at the level of the block group rather than individual blocks.

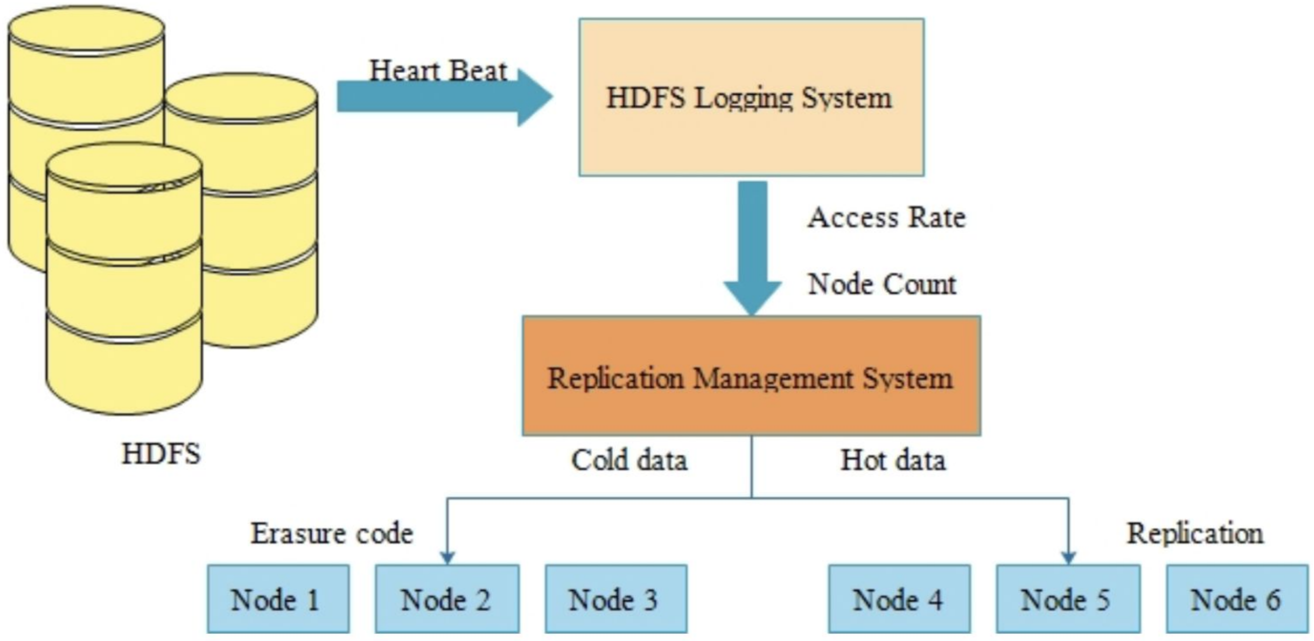
The second major change is the application of Erasure Code Policy. In HDFS-RAID, a whole system had to be encoded in only one EC Policy. The EC Policy is set in the configuration of Hadoop, and is applied as soon as Hadoop is started. In HDFS-EC of Hadoop 3.0, EC Policies are applied on files and directories. This allows different directories to have different EC Policies. Subsequently all files and subfolders within a folder on which EC policy has been applied, will also inherit that EC policy.

Another change is in the recovery process. Previously, recovery of loss blocks was done by the Namenode itself, however, in Hadoop 3.0, the Namenode is still responsible for block health monitoring using periodic heartbeats. Whenever the Namenode detects a failed block, it chooses an appropriate datanode to perform the recovery. The choice of the datanode is primarily based on locality of other related data blocks to minimize bandwidth requirements. Each datanode has an ECWorker [15] process running, which is responsible for recovery once contacted by Namenode.

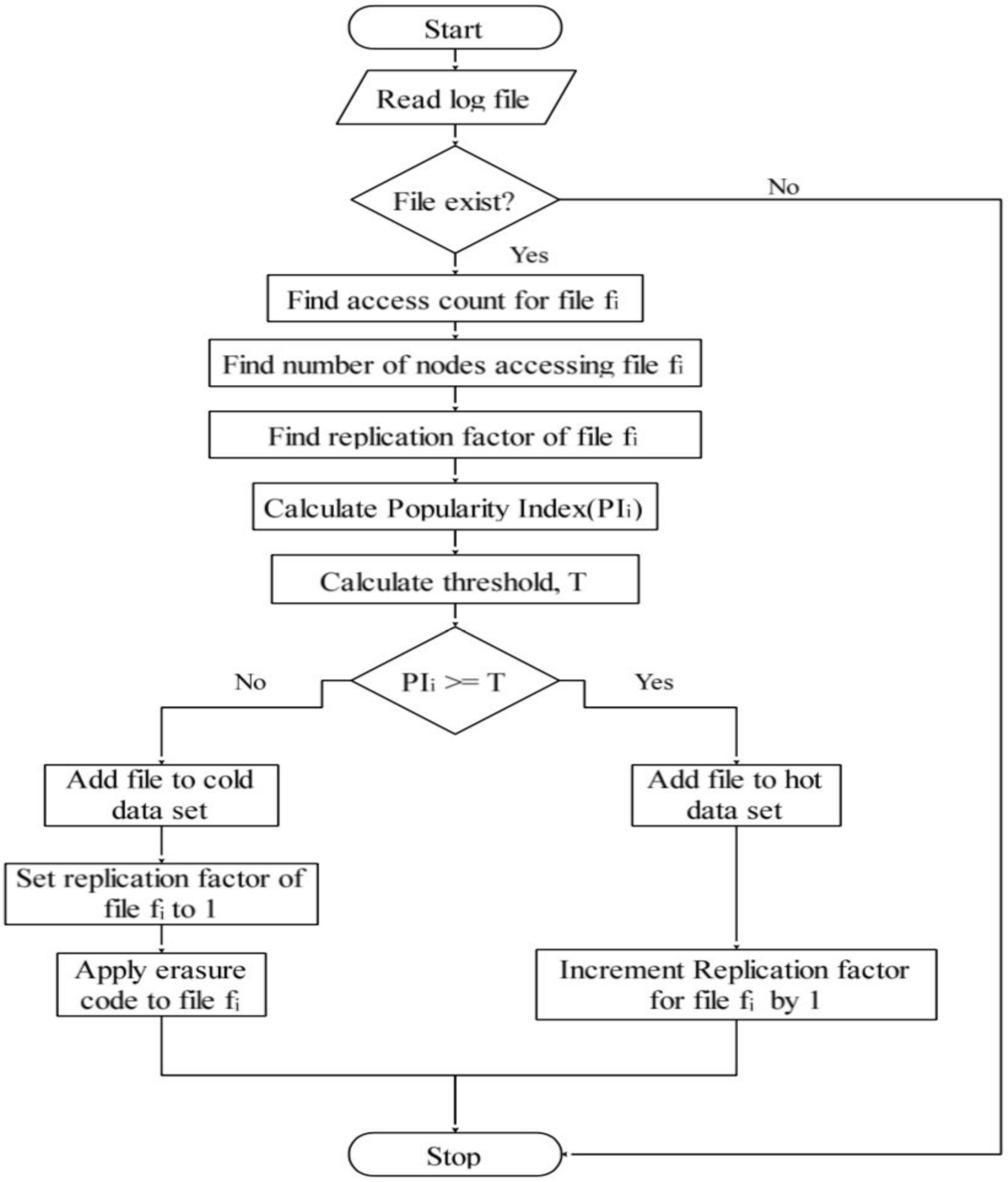
CHAPTER-3

PROPOSED WORK

3.1 ARCHITECTURE DIAGRAM



3.2 FLOWCHART


Start
Read log file
File exist?
No
Yes
Find access count for file f_i
Find number of nodes accessing file f_i
Find replication factor of file f_i
Calculate Popularity Index(PI_i)
Calculate threshold, T
$PI_i >= T$
No
Yes
Add file to cold data set
Set replication factor of file f_i to 1
Apply erasure code to file f_i
Add file to hot data set
Increment Replication factor for file f_i by 1
Stop

3.3 MODULAR DESCRIPTION:

3.3.1.SETTING UP SSH

Installation of Hadoop 3.2.1 is installed with the help of an installation tool called homebrew.

there are 3 different modes of Hadoop. We will only place our focus on using the Pseudo-distributed mode in this article.

1. Stand-alone mode
2. Pseudo-distributed mode
3. Distributed mode

we have checked whether ssh is enabled correctly using the command "ssh localhost".Then we have turned on the remote login .To enable SSH in MacOS, go to System Preference > Sharing, enable "Remote Login" and "Allow access for: All Users"

Prerequisite for hadoop installation is java version 8 since java version 9 is not supported in hadoop.

3.3.2.CONFIGURATION SETUP:

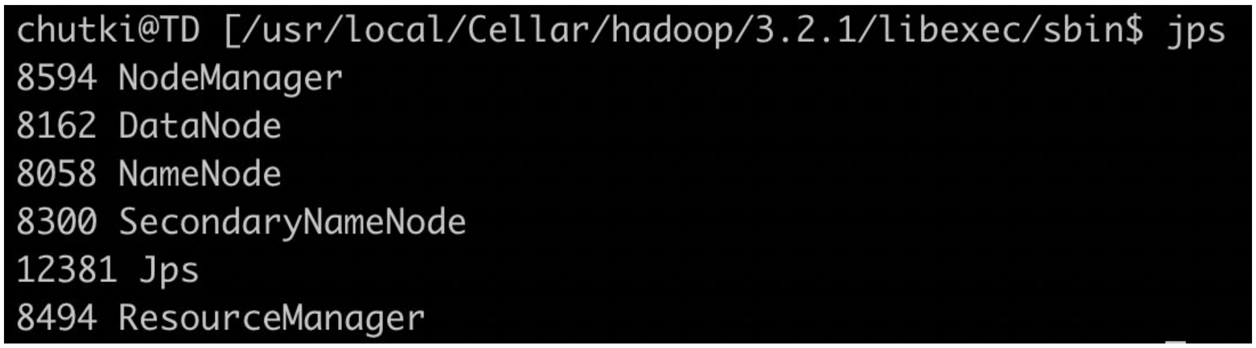
We need to create the following files for configuring namenodes , datanodes and replication factor .

1. hadoop-env.sh
2. core-site.xml

3. mapred-site.xml
4. hdfs-site.xml

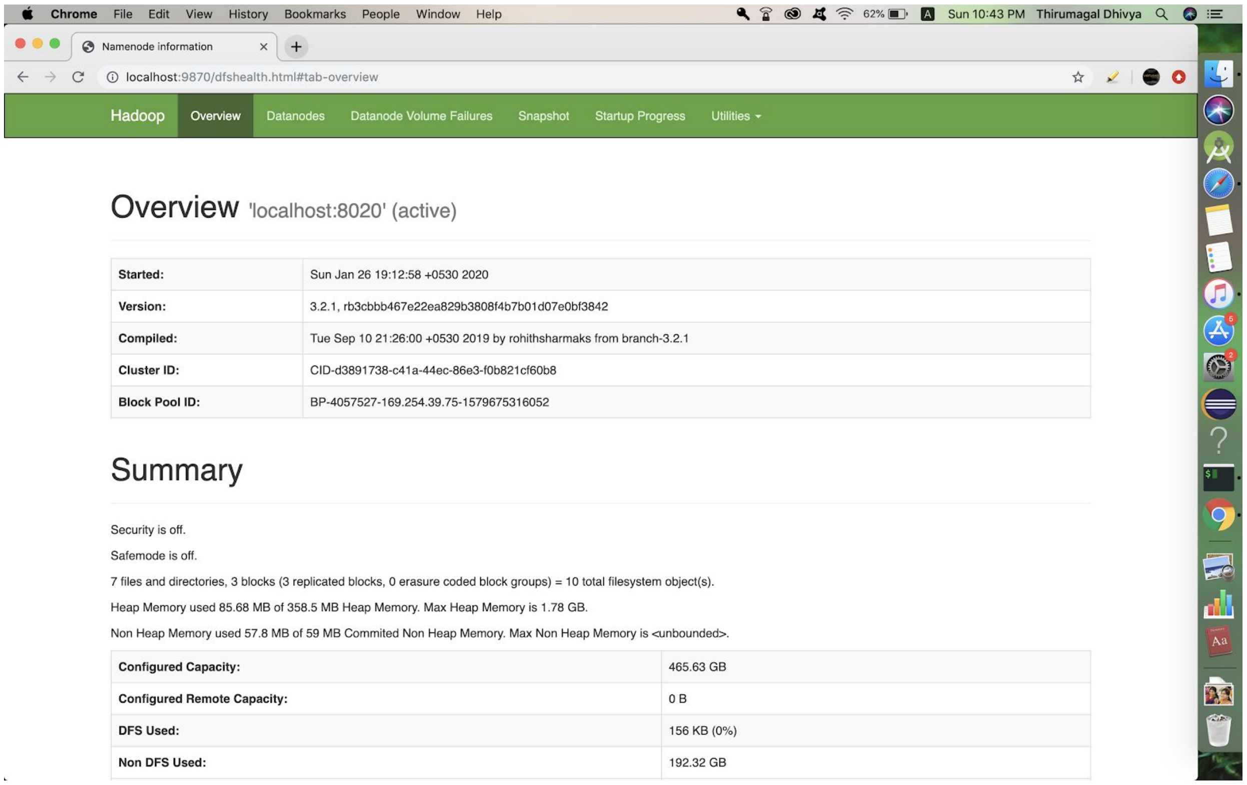
The location where the files and logs of namenode and datanode is stored is configured in the file core-site.xml and the port number for the localhost is set here. $JAVA_HOME path is set in the file hadoop-env.sh .Replication factor is set in the file hdfs-site.xml .By default , the replication factor is set as 3.In mapred-site.xml we will configure the jobtracker address and port number in map-reduce and a different port number is set to it. Finally, the last step before starting to launch the different services would be to format the HDFS.This is done using the command
"Hdfs namenode -format".

3.3.3.CLUSTER SETUP:



```
chutki@TD [/usr/local/Cellar/hadoop/3.2.1/libexec/sbin$ jps
8594 NodeManager
8162 DataNode
8058 NameNode
8300 SecondaryNameNode
12381 Jps
8494 ResourceManager
```

A single node cluster with namenodes and datanodes is thus created and we can check whether the services has been properly set up by using the java command called JPS
,then the running java process services will be listed .This cluster setup can also be viewed by the localhost http:/localhost:9870 , here the namenode,datanode information can be viewed.


Chrome File Edit View History Bookmarks People Window Help
62% Sun 10:43 PM Thirumagal Dhivya
Namenode information
localhost:9870/dfshealth.html#tab-overview
Hadoop
Overview
Datanodes
Datanode Volume Failures
Snapshot
Startup Progress
Utilities

Overview 'localhost:8020' (active)

Started:	Sun Jan 26 19:12:58 +0530 2020
Version:	3.2.1, rb3cbbb467e22ea829b3808f4b7b01d07e0bf3842
Compiled:	Tue Sep 10 21:26:00 +0530 2019 by rohithsharmaks from branch-3.2.1
Cluster ID:	CID-d3891738-c41a-44ec-86e3-f0b821cf60b8
Block Pool ID:	BP-4057527-169.254.39.75-1579675316052

Summary

Security is off.

Safemode is off.

7 files and directories, 3 blocks (3 replicated blocks, 0 erasure coded block groups) = 10 total filesystem object(s).

Heap Memory used 85.68 MB of 358.5 MB Heap Memory. Max Heap Memory is 1.78 GB.

Non Heap Memory used 57.8 MB of 59 MB Commited Non Heap Memory. Max Non Heap Memory is <unbounded>.

Configured Capacity:	465.63 GB
Configured Remote Capacity:	0 B
DFS Used:	156 KB (0%)
Non DFS Used:	192.32 GB

3.3.4.MAP REDUCE PROGRAM:

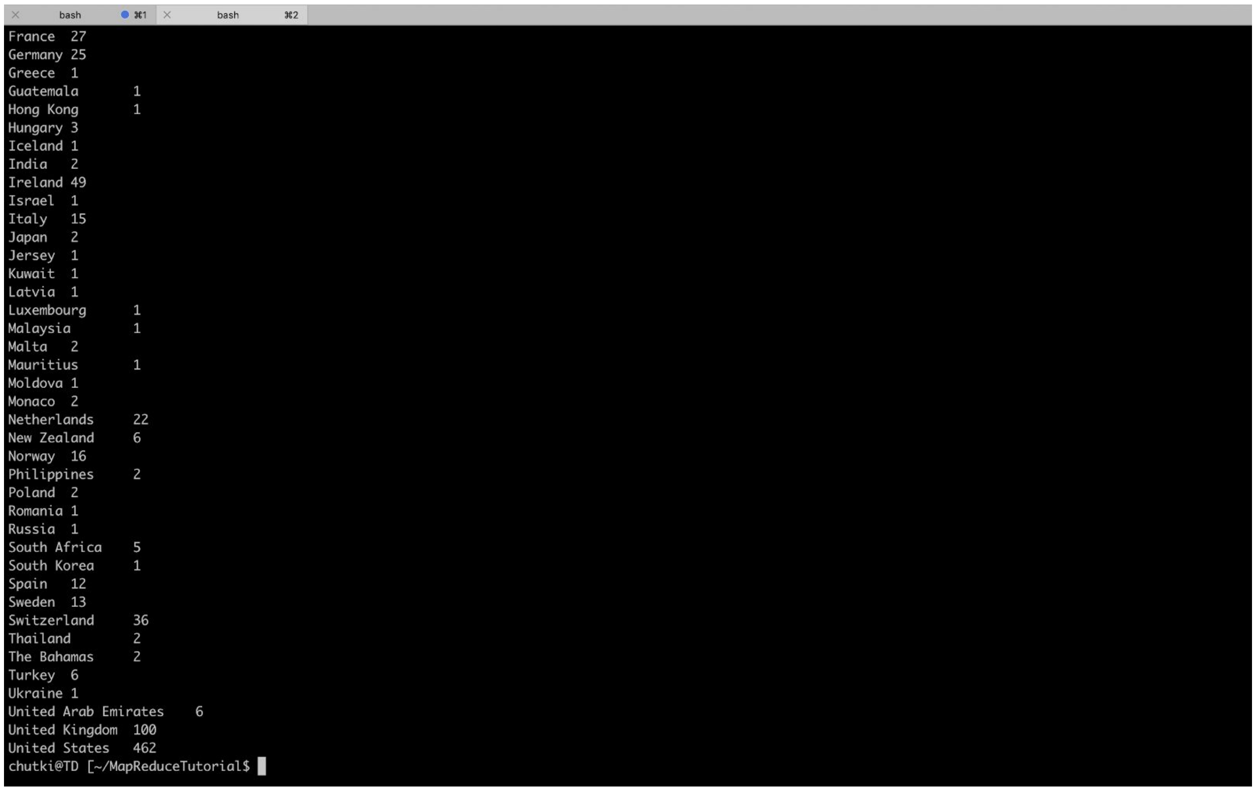
A map reduce program is executed to verify whether the mapping and reduce functions are working correctly . The input data used is SalesJan2009.csv. It contains Sales related information like Product name, price, payment mode, city, country of client etc. The goal is to ***Find out*** Number of Products Sold in Each Country.the three files in this program are,

SalesMapper.java

SalesCountryReducer.java

SalesCountryDriver.java

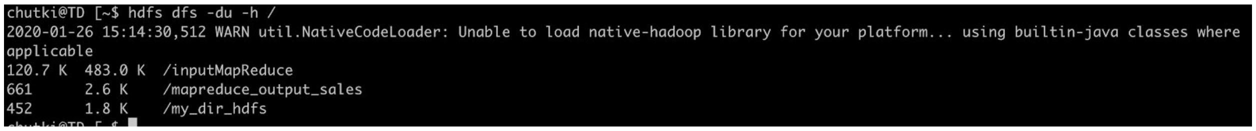
First the classpath is exported and java files are compiled , then the main class is defined in the manifest file .Then the jar file is created then the hadoop is started using the command start-all.sh.then the input file is copied into the local file system and then it is copied into the hadoop file system.Then the map reduce program is executed .Then the output file where the sales count for each country is stored in the hdfs file system.



```
France  27
Germany 25
Greece  1
Guatemala       1
Hong Kong       1
Hungary 3
Iceland 1
India   2
Ireland 49
Israel  1
Italy   15
Japan   2
Jersey  1
Kuwait  1
Latvia  1
Luxembourg      1
Malaysia        1
Malta   2
Mauritius       1
Moldova 1
Monaco  2
Netherlands     22
New Zealand     6
Norway  16
Philippines     2
Poland  2
Romania 1
Russia  1
South Africa    5
South Korea     1
Spain   12
Sweden  13
Switzerland     36
Thailand        2
The Bahamas     2
Turkey  6
Ukraine 1
United Arab Emirates    6
United Kingdom  100
United States   462
chutki@TD [~/MapReduceTutorial$
```

3.3.5. COPY PROGRAM USING JAVA API:

Using the java API ,the file in the local file system is copied to the hdfs file system and hdfs file system to the local file system.For this, the create() method creates the file in the hdfs file system and the file to copy is stored in the local file system and the copy program is executed after creating the jar file .This api uses one argument ,the path where the file has to be stored in hdfs .After executing this program ,the file gets stored in the hdfs file system which can be viewed by the command “hdfs dfs -ls /”.

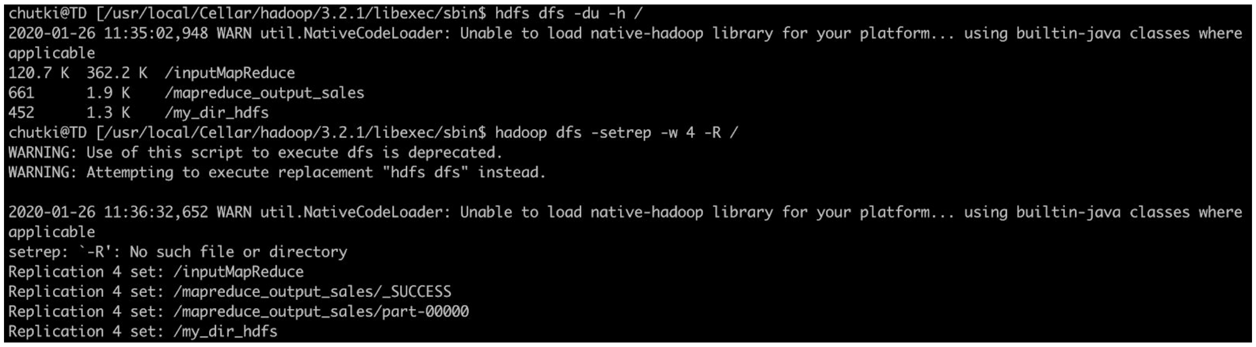


```
chutki@TD [~$ hdfs dfs -du -h /
2020-01-26 15:14:30,512 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where
applicable
120.7 K  483.0 K  /inputMapReduce
661      2.6 K    /mapreduce_output_sales
452      1.8 K    /my_dir_hdfs
chutki@TD [~$
```

my_dir _hdfs is the file that has been copied .

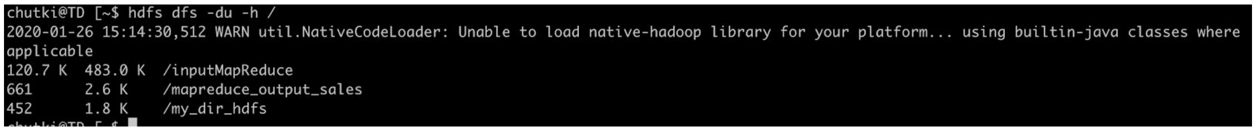
3.3.6. CHANGING REPLICATION FACTOR AND EVALUATION OF STORAGE ALLOCATION:

The replication factor is by default set as three , when it is set as three the file /input_map_reduce has 120 kb as its original size and the replication size will be 362 kb. This replication factor is changed into four ,



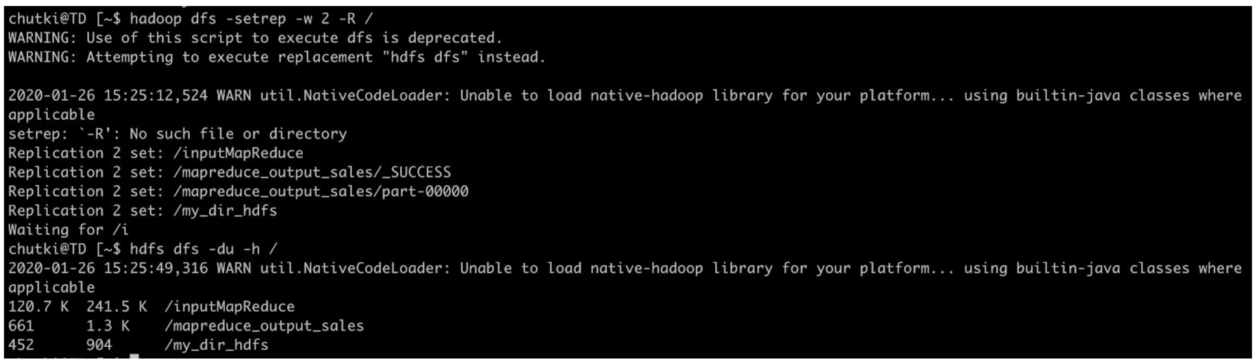
```
chutki@TD [/usr/local/Cellar/hadoop/3.2.1/libexec/sbin$ hdfs dfs -du -h /
2020-01-26 11:35:02,948 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where
applicable
120.7 K  362.2 K  /inputMapReduce
661      1.9 K    /mapreduce_output_sales
452      1.3 K    /my_dir_hdfs
chutki@TD [/usr/local/Cellar/hadoop/3.2.1/libexec/sbin$ hadoop dfs -setrep -w 4 -R /
WARNING: Use of this script to execute dfs is deprecated.
WARNING: Attempting to execute replacement "hdfs dfs" instead.

2020-01-26 11:36:32,652 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where
applicable
setrep: `-R': No such file or directory
Replication 4 set: /inputMapReduce
Replication 4 set: /mapreduce_output_sales/_SUCCESS
Replication 4 set: /mapreduce_output_sales/part-00000
Replication 4 set: /my_dir_hdfs
```

then the file size is 483 kb which is four times its original size (120 * 4)



```
chutki@TD [~$ hdfs dfs -du -h /
2020-01-26 15:14:30,512 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where
applicable
120.7 K  483.0 K  /inputMapReduce
661      2.6 K    /mapreduce_output_sales
452      1.8 K    /my_dir_hdfs
```

Then again the replication factor is set as two then the storage is calculated . Now the file size of /input_map_reduce is 240 kb which is two times its original size (120 * 2)



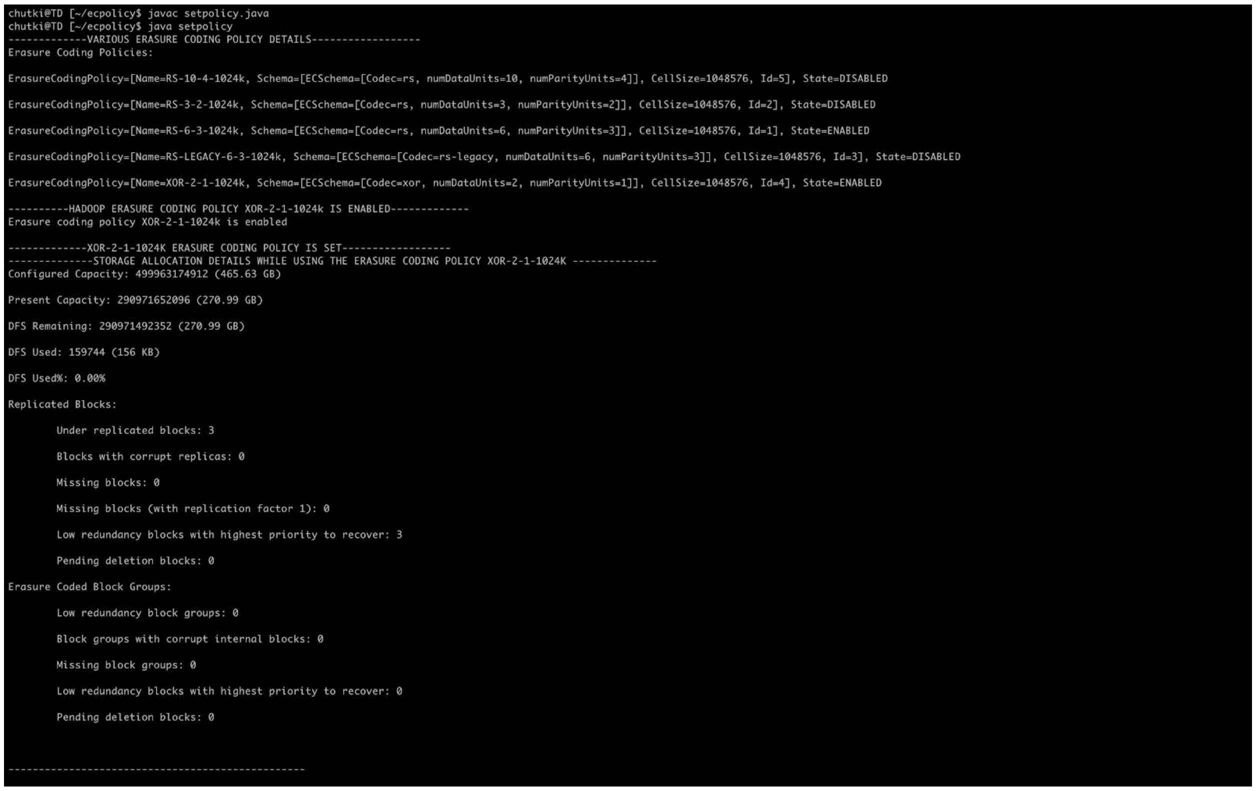
```
chutki@TD [~$ hadoop dfs -setrep -w 2 -R /
WARNING: Use of this script to execute dfs is deprecated.
WARNING: Attempting to execute replacement "hdfs dfs" instead.

2020-01-26 15:25:12,524 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where
applicable
setrep: `-R': No such file or directory
Replication 2 set: /inputMapReduce
Replication 2 set: /mapreduce_output_sales/_SUCCESS
Replication 2 set: /mapreduce_output_sales/part-00000
Replication 2 set: /my_dir_hdfs
Waiting for /i
chutki@TD [~$ hdfs dfs -du -h /
2020-01-26 15:25:49,316 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where
applicable
120.7 K  241.5 K  /inputMapReduce
661      1.3 K    /mapreduce_output_sales
452      904      /my_dir_hdfs
```

3.3.7.ERASURE CODING POLICY SET UP :

Various erasure coding policies are enabled and set up and the performance is evaluated .The various erasure coding policies are ,

- RS -3-2-1024K
- RS-6-3-1024K
- RS-10-4-1024K
- XOR-3-2-1024K

These erasure coding policies are enabled and the performance is measured .The storage allocation while using various erasure coding policies is calculated and based on that the erasure coding policy is allocated dynamically.

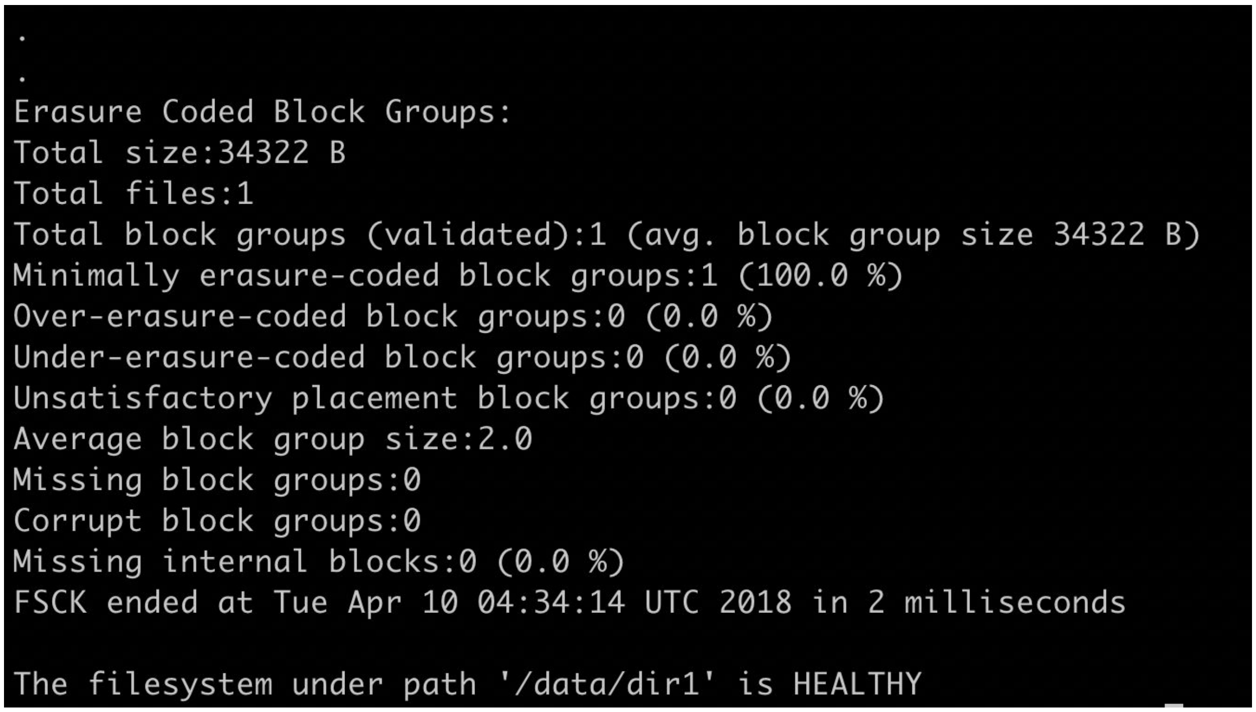


```
chutki@TD [~/ecpolicy$ javac setpolicy.java
chutki@TD [~/ecpolicy$ java setpolicy
-------------VARIOUS ERASURE CODING POLICY DETAILS------------------
Erasure Coding Policies:

ErasureCodingPolicy=[Name=RS-10-4-1024k, Schema=[ECSchema=[Codec=rs, numDataUnits=10, numParityUnits=4]], CellSize=1048576, Id=5], State=DISABLED

ErasureCodingPolicy=[Name=RS-3-2-1024k, Schema=[ECSchema=[Codec=rs, numDataUnits=3, numParityUnits=2]], CellSize=1048576, Id=2], State=DISABLED

ErasureCodingPolicy=[Name=RS-6-3-1024k, Schema=[ECSchema=[Codec=rs, numDataUnits=6, numParityUnits=3]], CellSize=1048576, Id=1], State=ENABLED

ErasureCodingPolicy=[Name=RS-LEGACY-6-3-1024k, Schema=[ECSchema=[Codec=rs-legacy, numDataUnits=6, numParityUnits=3]], CellSize=1048576, Id=3], State=DISABLED

ErasureCodingPolicy=[Name=XOR-2-1-1024k, Schema=[ECSchema=[Codec=xor, numDataUnits=2, numParityUnits=1]], CellSize=1048576, Id=4], State=ENABLED

----------HADOOP ERASURE CODING POLICY XOR-2-1-1024k IS ENABLED-------------
Erasure coding policy XOR-2-1-1024k is enabled

-------------XOR-2-1-1024K ERASURE CODING POLICY IS SET------------------
--------------STORAGE ALLOCATION DETAILS WHILE USING THE ERASURE CODING POLICY XOR-2-1-1024K --------------
Configured Capacity: 499963174912 (465.63 GB)

Present Capacity: 290971652096 (270.99 GB)

DFS Remaining: 290971492352 (270.99 GB)

DFS Used: 159744 (156 KB)

DFS Used%: 0.00%

Replicated Blocks:

        Under replicated blocks: 3

        Blocks with corrupt replicas: 0

        Missing blocks: 0

        Missing blocks (with replication factor 1): 0

        Low redundancy blocks with highest priority to recover: 3

        Pending deletion blocks: 0

Erasure Coded Block Groups:

        Low redundancy block groups: 0

        Block groups with corrupt internal blocks: 0

        Missing block groups: 0

        Low redundancy blocks with highest priority to recover: 0

        Pending deletion blocks: 0


-------------------------------------------------
```

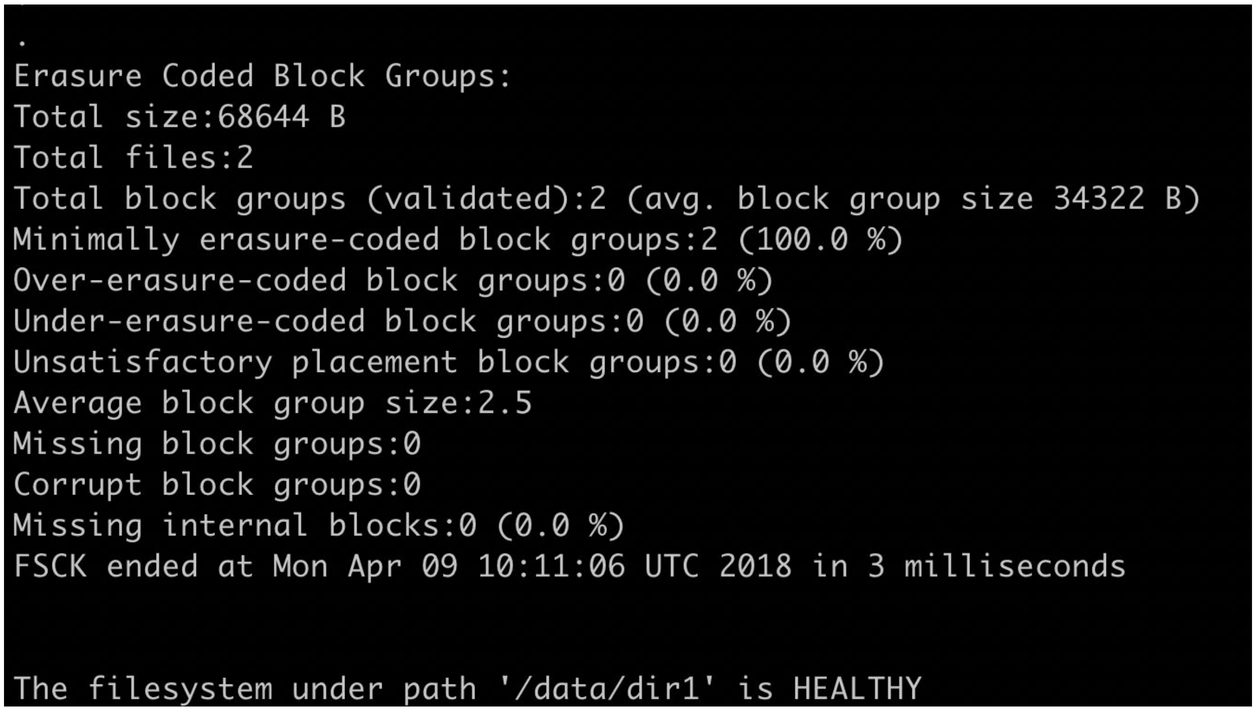
3.3.8.DECPA :

The file with size 34322B ,while we set the erasure coding policy using the decpa algorithm , RS -6-2 is set and the average block size is 2 .That is ,two stripes is needed to store the file , which is best than the erasure policy RS-10-2 where the block size will be 1280 .

Let us consider another example ,where the file size is 68644B , while we set the erasure coding policy using the decpa algorithm, RS-3-2 and the average block size is 2.5 ,using this policy is efficient in respect to storage than for other erasure coding policies .



```
.
.
Erasure Coded Block Groups:
Total size:34322 B
Total files:1
Total block groups (validated):1 (avg. block group size 34322 B)
Minimally erasure-coded block groups:1 (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 %)
Average block group size:2.0
Missing block groups:0
Corrupt block groups:0
Missing internal blocks:0 (0.0 %)
FSCK ended at Tue Apr 10 04:34:14 UTC 2018 in 2 milliseconds

The filesystem under path '/data/dir1' is HEALTHY
```



```
.
Erasure Coded Block Groups:
Total size:68644 B
Total files:2
Total block groups (validated):2 (avg. block group size 34322 B)
Minimally erasure-coded block groups:2 (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 %)
Average block group size:2.5
Missing block groups:0
Corrupt block groups:0
Missing internal blocks:0 (0.0 %)
FSCK ended at Mon Apr 09 10:11:06 UTC 2018 in 3 milliseconds


The filesystem under path '/data/dir1' is HEALTHY
```

3.3.9.REPLICATION FACTOR SET UP DYNAMICALLY:

The replication factor has been set dynamically by determining the popularity index of the file ,determining the file whether it is a hot data or cold data by determining the access count of the file by finding the number of nodes accessing the file using the log file that is found inside the current folder. Popularity index is calculated using the formula PIi = (aci * nci) / rfi . Setting a low replication factor for least accessed file and setting a high replication factor for frequently accessed file will improve the storage consequently.

CHAPTER-4

IMPLEMENTATION AND ANALYSIS

4.1 PROJECT DESCRIPTION

DECPA – DYNAMIC ERASURE CODING POLICY ALLOCATION

The default block size of Hadoop 3.0 has increased from 64 to 128 MB. Assuming that RS(3,2) EC Policy is used, any file with file size less than 384 (128*3) MB will fit within a stripe of 3 data blocks. However if the file size is greater than 384, then 2 stripes of RS(3,2) will be required. Alternatively, the policy RS(6,3) can be used, which can store up 768 MB in a single stripe. And finally if the file/folder size is greater than 768 MB, then RS(10,4) can also be used. So as to reap the benefits of the different EPolicies, new EC policies {RS(4,3), RS(5,3), RS(7,3), RS(7,4), RS(8,4), RS(9,4)} were implemented, and then DECPA was designed. DECPA will allocate EC policies so as to minimize the number of stripes produced, hence minimize storage usage. The efficiency (amount of extra storage vs number of failures tolerable) of the EC Policies .

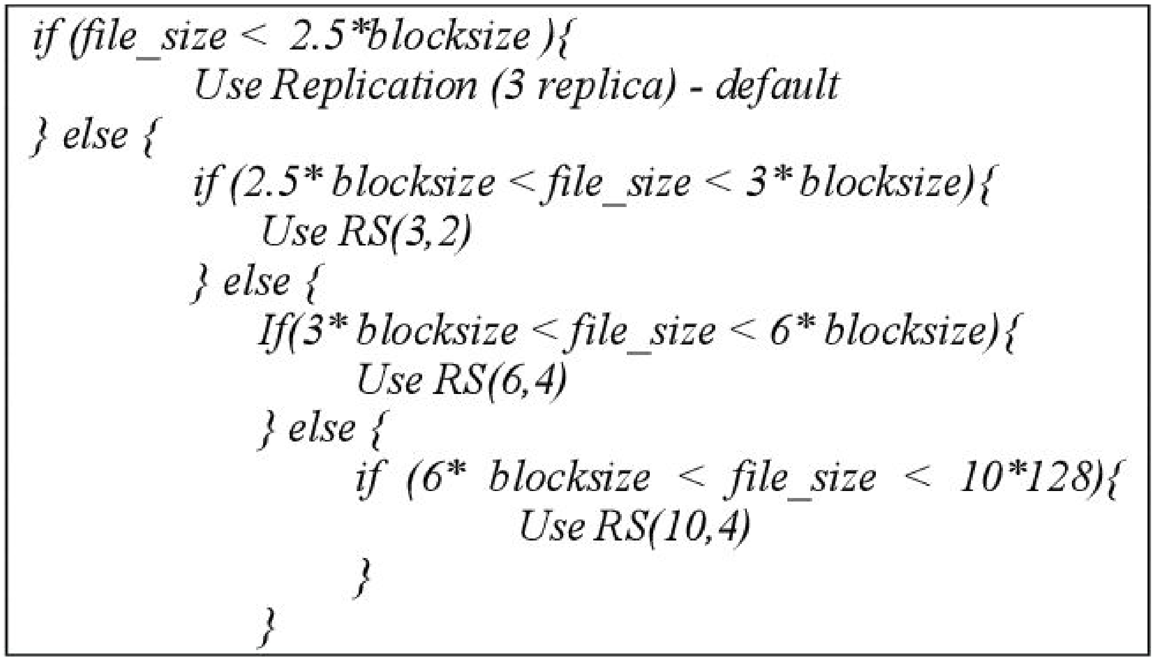
From the graph, it can be seen that the maximum efficiency is reaped from RS(7,3) with an average size stripe (7*128 MB plus 3*128MB). However if a file or a number of files grouped as one with size 600MB for example, will result in the creation of a number of padded blocks (blank ones) to create a complete stripe and that is the rationale for coming up with DECPA.

A. Algorithm of DECPA

The default block size of Hadoop 3.0 has increased from 64 to 128 MB. Assuming that RS(3,2) EC Policy is used, any file with file size less than 384 (128*3) MB will fit within a stripe of 3 data blocks. However if the file size is greater than 384, then 2 stripes of RS(3,2) will be required. Alternatively, the policy RS(6,3) can be used, which can store up 768 MB in a single stripe. And finally if the file/folder size is greater than 768 MB, then RS(10,4) can also be used. So as to reap the benefits of the different EPolicies, new EC policies {RS(4,3), RS(5,3), RS(7,3), RS(7,4), RS(8,4), RS(9,4)} were implemented, and then DECPA was designed. DECPA will allocate EC policies so as to minimize the number of stripes produced, hence minimize storage

By default, the Block Size in Hadoop 3.0 is 128 MB and each erasure code policy will have m data blocks and n parity blocks. We consequently derive an algorithm that would be used for determining the optimal policy to be applied for any given folder in terms of minimising storage overhead.

Given that RS(10,4) is the policy that can hold the most data, we present the first of DECPA-F whereby it considers folders with less than 10 data blocks * 128 MB (size of one block), that is a total 1280 MB.



```
if (file_size <  2.5*blocksize ){
            Use Replication (3 replica) - default
} else {
            if (2.5* blocksize < file_size < 3* blocksize){
                Use RS(3,2)
            } else {
                If(3* blocksize < file_size < 6* blocksize){
                     Use RS(6,4)
                } else {
                     if  (6*  blocksize  <  file_size  <  10*128){
                              Use RS(10,4)
                     }
                }
```

Fig 3 – DECPA-F

The algorithm DECPA-F, is used if an object is of the size less than 1280 MB, that is will fit in 10 blocks or less. For file sizes less than 320 MB, simple 3-way Replication is recommended as it is the default storage redundancy method in Hadoop 3.0.0. For other file sizes, EC policies are applied as per DECPA-F. In Figure 3, only the 3 default EC policies are shown. However, since we have also implemented RS(4,3), RS(5,3), RS(7,4), RS(8,4)and RS(9,4), they are also incorporated in the DECPA-F algorithm.

<u>Algorithm of DECPA-M (Multiple)</u>

The algorithm DECPA-M, is used if an object is of the size more than 1280 MB, that is, it will require more than one stripe. In DECPA-M, the amount of overhead that will be produced, is used as criterion, to apply the required EC policy. The optimal EC policy is chosen based on minimal overhead generated.

```
if (file_size > 10* blocksize) {
        //find RS that provides smallest storage overhead
        overhead = 0;
        For (n=5, n++, n<=14){
            exclude n = 6;
            n_overhead = modulus(n* blocksize, file_size)
            if(n_overhead < overhead){
                overhead=n_overhead;
                no_of_block=n;
            }
        }
        Apply RS(n,m);
}
```

C. Algorithm of DECPA-MF (Multiple RS(10,4) and 1 DECPA-F)

```
if (file_size > 10*128) {

//use RS(10,4) for x stripes n=10

}

stripes = modulus(n*128, file_size) for(stripes){

Apply RS(10,4) }

remaining = file_size – stripes*(n*128) Apply DECPA-F using remaining;

}
```

The algorithm DECPA-MF, is used if an object is of the size more than 1280 MB, that is it will require more than one stripe. In DECPA-MF, the max amount of RS(10,4) stripes are created, and the remaining is encoded using the DECPA- F algorithm.

<u>DYNAMIC DATA REPLICATION</u>

The algorithm divides each data into two categories as hot data or cold data based on their popularity in the Hadoop cluster. To determine the popularity of a file, different parameters like number of accesses to that file, number of nodes accessing the file and the replication factor of the file are considered. To obtain these values the log files in HDFS are analyzed. After analyzing and extracting the required values the popularity for each file is calculated. The Popularity Index (PIi) of a file fi, is calculated as follows:

PIi = (aci * nci) / rfi

where, aci is the number of access received to a file fi, nci denotes the number of nodes which accessed fi and rfi represents the current replication factor of fi in HDFS. After calculating the popularity of each file, a threshold value, T, is obtained by calculating the mean of popularity values as follows:

The value of T is used to classify the files in HDFS. Popularity Index of each file is compared with the obtained threshold value. The file with popularity greater than or equal to T is classified as hot and the file with popularity less than threshold is classified as cold. The popularity is comparatively low for the files classified as cold. Maintaining three copies for these least popular files is wastage of space. So the replication factor of these files is set to 1. This change in replication count will make these files vulnerable for threats like data loss and unavailability. To overcome this risk and to provide data availability, the concept of erasure code is implemented. Reed-Solomon (10,4) erasure coding is applied to cold files. Reed- Solomon (10,4) divides a file into 10 equally sized blocks with 6 data blocks and 4 parity blocks. All the files classified as cold are encoded using Reed-Solomon. The above process is iterative and depending on the popularity of data over time, a data file will move from hot data to cold data and vice versa.

The files in the hot category are the files with high popularity. These files will be accessed more than the cold files. So it is important to maintain more than one copy for these files. By providing more copies for the popular files, computation performance of the Hadoop system can be improved. In order to ensure the availability and to reduce the overall performance time, an additional copy of the popular file is created by incrementing the replication factor of the hot file by one.

Proposed Dynamic Data Replication Algorithm

```
Input: log
Begin
1.   Set time interval
2.   For each time interval
     {
       i. read logfile
      ii. for each file $f_i$
          {
              a.   Find $ac_i$, $nc_i$, $rf_i$
              b.   Calculate popularity index($PI_i$) of each
                   file
                           $PI_i = (ac_i * nc_i) / rf_i$
          }
     iii. Calculate the threshold,
                           $T = \frac{\sum_{i=1}^{n} PI_i}{n}$
     iv. For each file $f_i$
              Compare threshold T
                   If $PI_i >= T$
                           $hd \leftarrow f_i$
                   Else
                           $cd \leftarrow f_i$
     v. For each $f_i$ in hd,
                   Increment $rf_i$ by 1.
     vi. For each $f_i$ in cd
                   Set $rf_i$ to1
                   Encode $f_i$ using Reed-Solomon erasure
                   code
     }end for
End
```

4.2.SYSTEM REQUIREMENTS:

A hadoop project can be done in

- Windows
- Mac
- linux

4.2.1.HARDWARE REQUIREMENTS:

- Minimum RAM required: 4GB (Suggested: 8GB)
- Minimum Free Disk Space: 25GB
- Minimum Processor i3 or above
- Operating System of 64bit (Suggested)

4.2.2.SOFTWARE REQUIREMENTS:

Java :

• The widely used and highly popular Java programming language is proved to be a great tool for developing platform independent applications.

• Java is considered to be blind friendly language

• In PC , java programming language is used to develop application

Hadoop 3.2.1 :

- Apache Hadoop is a collection of open-source software utilities that facilitate using a network of many computers to solve problems involving massive amounts of data and computation.

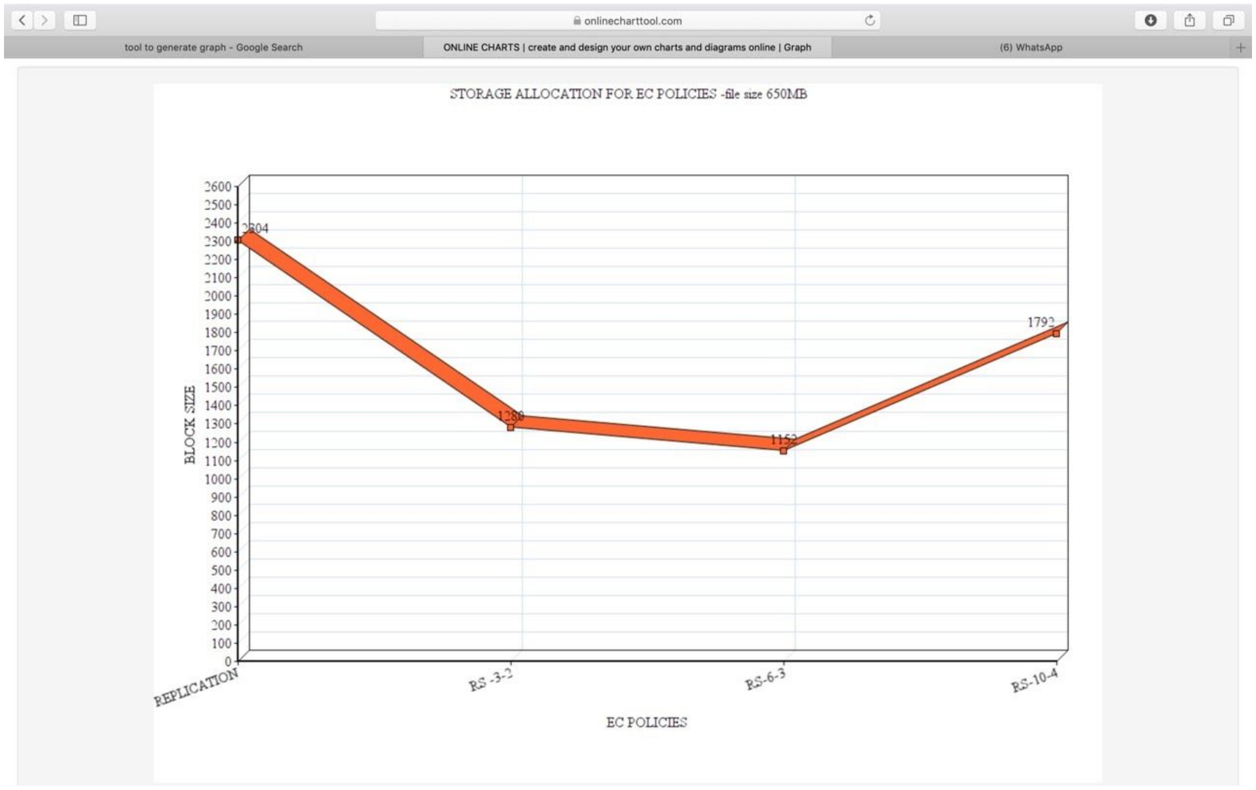
- It provides a software framework for distributed storage and processing of big data using the MapReduce programming model.

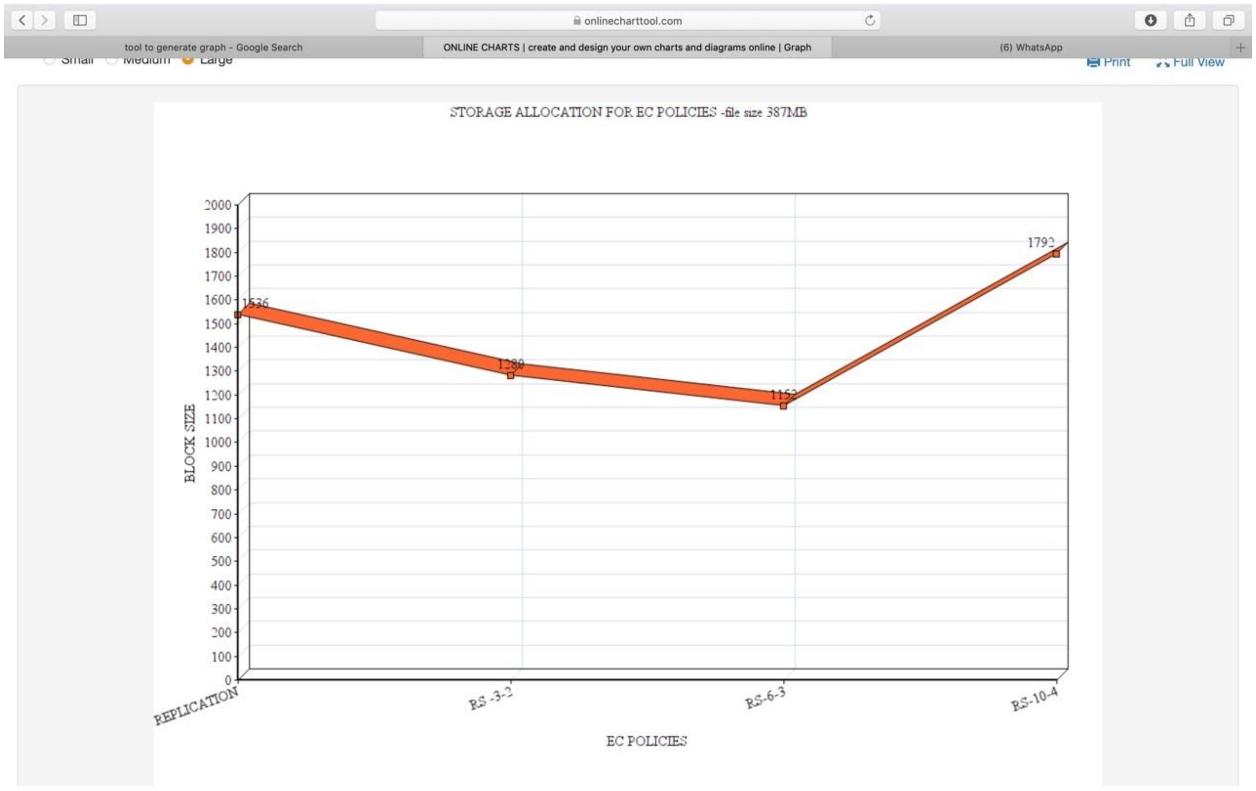
4.3 RESULT ANALYSIS

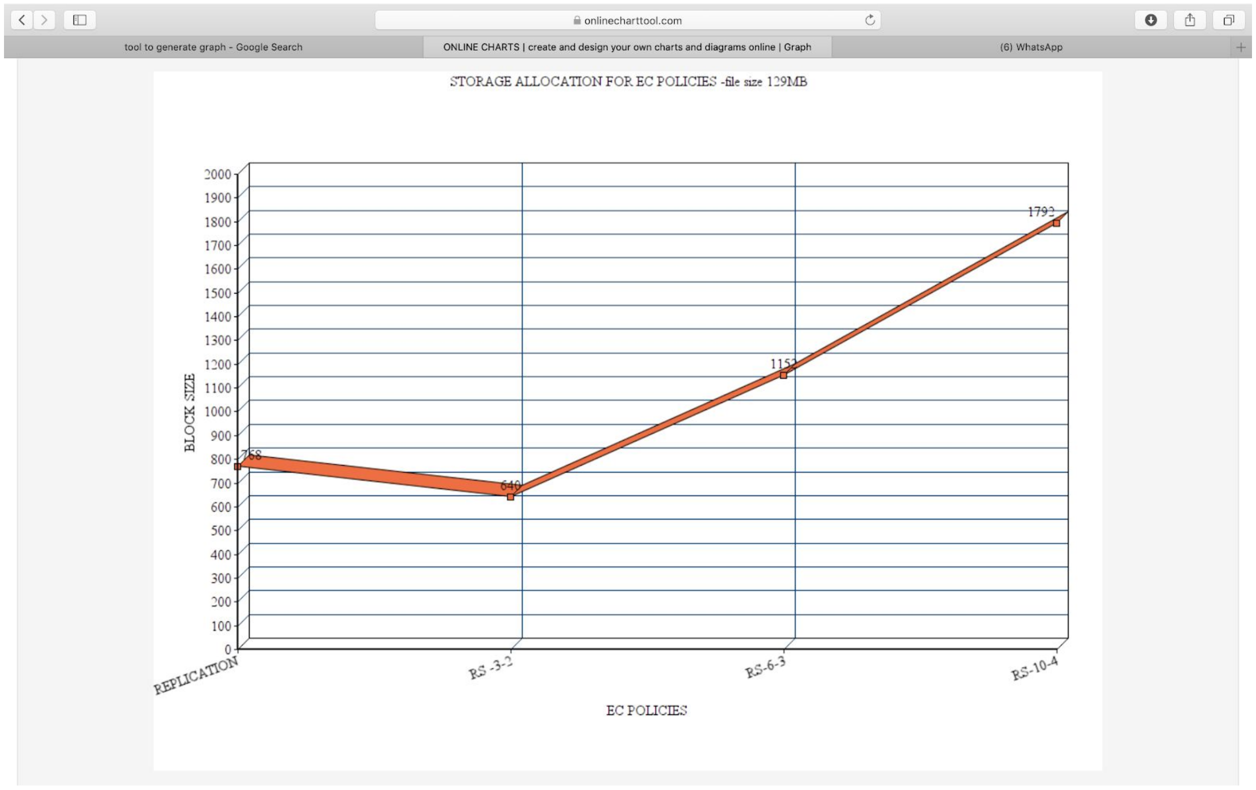
For the purpose of experimentation, a file size of 1.07 GB is used so as to generate enough blocks and stripes. Firsty the performances of writing the 1.07 Gb file to cluster under different policies are presented. Secondly the repair performances to simulate one and two node/block failures are presented . And finally the effectiveness of DECPA is evaluated by comparing it to the default Hadoop redundancy implementation.

A. Write Performances

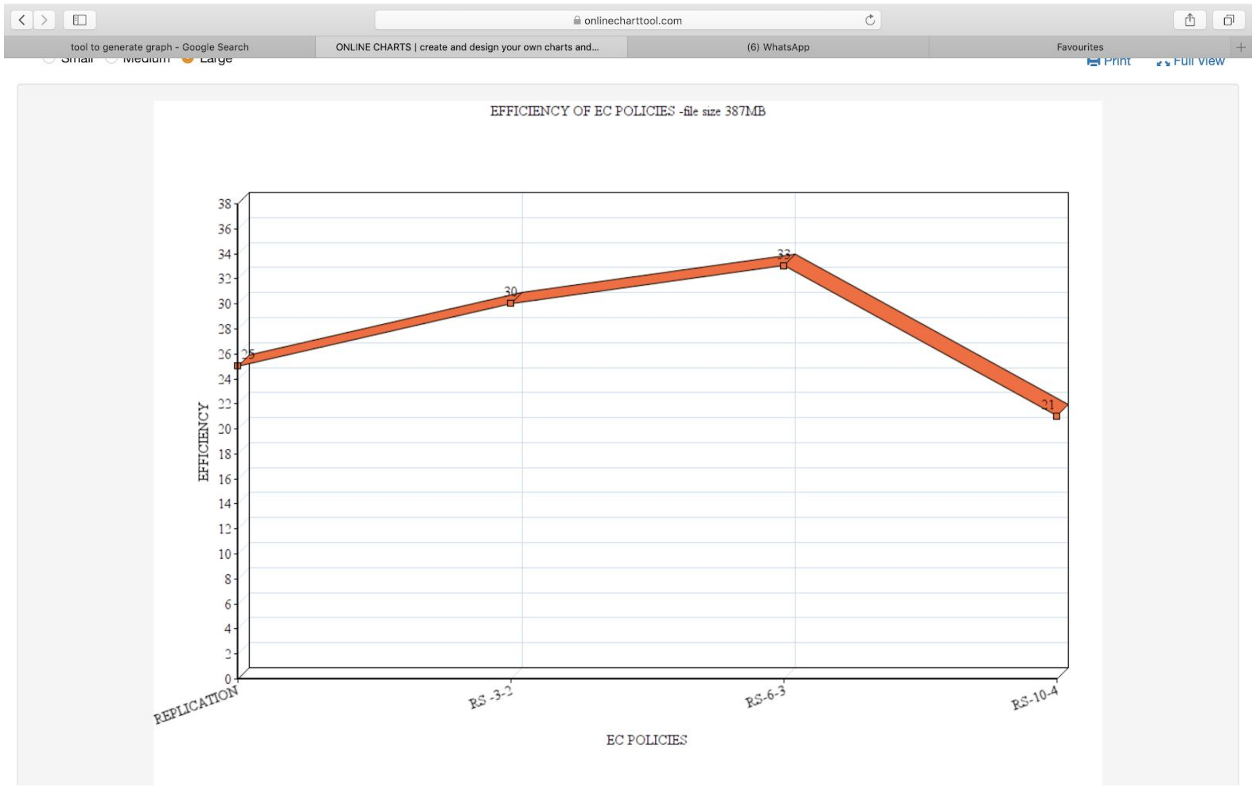
As mentioned for testing purposes, a file of the size of 1.07 GB was used to evaluate the performance. Figure 6 shows the time taken to write each stripe for the different EC policies used. Since RS(7,3) is the most balanced policy in terms of blocks and parity created, it provides the minimum time for writing the same amount of data to the cluster.


onlinecharttool.com
tool to generate graph - Google Search
ONLINE CHARTS | create and design your own charts and diagrams online | Graph
(6) WhatsApp
STORAGE ALLOCATION FOR EC POLICIES -file size 650MB
BLOCK SIZE
EC POLICIES
REPLICATION
RS -3-2
RS-6-3
RS-10-4
2304
1280
1152
1792


onlinecharttool.com
tool to generate graph - Google Search
ONLINE CHARTS | create and design your own charts and diagrams online | Graph
(6) WhatsApp
Small
Medium
Large
Print
Full View
STORAGE ALLOCATION FOR EC POLICIES -file size 387MB
BLOCK SIZE
EC POLICIES
REPLICATION
RS -3-2
RS-6-3
RS-10-4
1536
1280
1152
1792



onlinecharttool.com
tool to generate graph - Google Search
ONLINE CHARTS | create and design your own charts and diagrams online | Graph
(6) WhatsApp
STORAGE ALLOCATION FOR EC POLICIES -file size 129MB
BLOCK SIZE
2000
1900
1800
1700
1600
1500
1400
1300
1200
1100
1000
900
800
700
600
500
400
300
200
100
0
768
640
1152
1792
REPLICATION
RS -3-2
RS-6-3
RS-10-4
EC POLICIES



onlinecharttool.com
tool to generate graph - Google Search
ONLINE CHARTS | create and design your own charts and...
(6) WhatsApp
Favourites
Print
Full View
EFFICIENCY OF EC POLICIES -file size 387MB
EFFICIENCY
38
36
34
32
30
28
26
24
22
20
18
16
14
12
10
8
6
4
2
0
25
30
33
21
REPLICATION
RS -3-2
RS-6-3
RS-10-4
EC POLICIES

Storage allocation is calculated for various file sizes and block sizes and the efficiency is computed based on that.

B. Repair Performance

The repair performance, as shown in Figure 7, demonstrates an exponential trend with longer stripes requiring more time. This is due to the number of blocks transmission required for the repair of 1 or more failures. Given the limited number of nodes (12), we could only perform testing for 2 repairs using the RS(7,4) EC Policy.

As mentioned earlier, DECPA-F is used where the size of data can fit in a single stripe, that is less or equal to 1280 MB. From Figure 8, it can be clearly observed that DECPA-F will generate the least amount of the overhead for any EC policies used. And when the size of the data is 1280 MB, the storage saving can be as much as 4000MB.

DECPA-M is used when multiple stripes are indeed to encode the data, that is, more than 10 blocks will be generated. It can be observed from figure 9, that DECPA-M produces the least overheads irrespective of the EC policy used or the size of data.

DECPA-MF adopts a different approach, that of using RS(10,4) stripes to cover maximum data to be encoded, and the remainder is then encoded using DECPA-F, to find the appropriate EC policy producing least overheads. Though, from figure 10, DECPA-MF achieves lesser overheads compared to DECPA-M with greater data sizes, it requires more metadata and processing. In order to implement DECPA-MF, stripes encoded with RS(10,4) are stored and EC policy is applied to it, and the rest is stored and encoded using RS(m,n) based on DECPA-F, in a separate folder. It is

implemented in this way because EC policies in Hadoop 3.0 can only be applied to folders and not individual files.

<u>CHAPTER-5</u>

CONCLUSION

In this work, we designed and implemented DECPA, Dynamic Erasure Coding Policy Allocation, based on minimum overhead produced, in order to maximize storage capacity. Three types of allocation has been implemented, (i) DECPA-F, for file sizes less than 10 blocks, (ii) DECPA-M, for file sizes greater than 10 blocks using 1 EC policy and finally (iii) DECPA-MF, for file sizes greater than 10 blocks, using maximum RS(10,4) and RS(m.n) based on DECPA-F. We demonstrated the effectiveness of the algorithm developed. Though the savings might seem minimal (few GBs), however when the algorithm would be deployed globally, it would have a greater impact.HDFS is equipped with a mechanism that uniformly replicates every file without considering the popularity of the file. However, this replication strategy still remains a critical drawback with regards to the storage aspect. To overcome this drawback a storage efficient dynamic data replication strategy is implemented which can dynamically adapt to changes in data popularity.The storage space is optimized by reducing replication factor of the cold files and applying erasure code. Application of this strategy results in substantial storage-cost savings in hardware expenditure. This work can be improved further by optimizing the placement strategy of replicas in Hadoop cluster. Also a comparison can be made based on the performance of proposed strategy and existing method in Hadoop.

CHAPTER-6

REFERENCES

[1] Hadoop.apache.org. (2018). Apache Hadoop 3.0.0 – HDFS Erasure Coding. [online] Available at: https://hadoop.apache.org/docs/r3.0.0/hadoop-project-dist/hadoop-hdfs/HDFSErasureCoding.html [Accessed 24 Sep. 2018].

[2] Zhe, Z., Wang, A., Zheng, K., Maheswara, U. and Vinaya, k. (2018). Introduction HDFS Erasure Coding in Apache Hadoop - Cloudera Engineering Blog. [online] Cloudera Engineering Blog. Available at: http://blog.cloudera.com/blog/2015/09/introduction-to-hdfs-erasure-coding-in-apache-hadoop/ [Accessed 24 Sep. 2018].

[3] Chiniah, A., Dhora, J.A.D. and Sandooram, C.J., 2017, May. Erasure-Coded Network Backup System (ECNBS). In International Conference on Information, Communication and Computing Technology (pp. 35-43). Springer, Singapore.

[4] Li, J. and Li, B., 2016, June. Zebra, 2016: Demand-aware erasure coding for distributed storage systems. In 2016 IEEE/ACM 24th International Symposium on Quality of Service (IWQoS) (pp. 1-10).

[5] Meng, Y., Zhang, L., Xu, D., Guan, Z. and Ren, L., 2019, February. A Dynamic Erasure Code Based on Block Code. In Proceedings of the 2019 International Conference on Embedded Wireless Systems and Networks (pp. 379-383). Junction Publishing.

[6] Lee, C.W., Hsieh, K.Y., Hsieh, S.Y. and Hsiao, H.C., 2014. A dynamic data placement strategy for hadoop in heterogeneous environments. Big Data Research, 1, pp.14-22.

[7] Shen, J., Li, Y., Sheng, G., Zhou, Y. and Wang, X., 2018, October. Efficient Memory Caching for Erasure Coding Based Key-Value Storage Systems. In CCF Conference on Big Data (pp. 512-539). Springer, Singapore.

[8] Chen et. al. "Giza: Erasure Coding Objects across Global Data Centers", USENIX ATC 2017

[9] Wei, Q., Veeravalli, B., Gong, B., Zeng, L. and Feng, D., 2010, September. CDRM: A cost-effective dynamic replication management scheme for cloud storage cluster. In Cluster Computing (CLUSTER), 2010 IEEE International Conference on (pp. 188-196). IEEE.

[10] Cheng, Z., Luan, Z., Meng, Y., Xu, Y., Qian, D., Roy, A., Zhang, N. and Guan, G., 2012, September. Erms: An elastic replication management system for hdfs. In Cluster Computing Workshops (CLUSTER WORKSHOPS), 2012 IEEE International Conference on (pp. 32-40). IEEE.

[11] Masur, R.G. and Mcintosh, S.K., 2017, August. Preliminary performance analysis of Hadoop 3.0. 0-alpha3. In Scientific Data Summit (NYSDS), 2017 New York (pp. 1-3). IEEE.

[12] Shubham, S. (2018). Hadoop 3 | What's New in Hadoop 3.0 | Hadoop 3 Enhancements | Edureka. [online] Edureka Blog. Available at: https://www.edureka.co/blog/hadoop-3/ [Accessed 24 Sep. 2018].

[13] Hadoop.apache.org. (2018). Apache Hadoop 2.9.1 – Rack Awareness. [online] Available at: https://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-common/RackAwareness.html [Accessed 23 Sep. 2018].

[14] Apache (2018). Apache Hadoop 3.0.0-alpha1-SNAPSHOT – HDFS Erasure Coding. [online] Iwasakims.github.io. Available at: http://iwasakims.github.io/HDFS-9884/hadoop-project/hadoop-project-dist/hadoop- hdfs/HDFSErasureCoding.html [Accessed 3 Oct. 2018].

[15] Chiniah, A., Esmaili, K.S. and Datta, A., 2013, October. Efficient updates in cross-object erasure-coded storage systems. In 2013 IEEE International Conference on Big Data (pp. 28-32). IEEE.

[16] Chiniah A. and Einstein M.U.A. (2019) HIVE-EC: Erasure Code Functionality in HIVE Through Archiving. In: Advances in Information and Communication Networks. FICC 2018. Advances in Intelligent Systems and Computing, vol 887. Springer,Cham.