Shanghai Jiaotong University

Big Data Processing Technology

Project 1: Erasure Code in Hadoop

Author:
FEI Yixiao
118260910031
LI Yanhao
118260910036
LUO Dihao
118260910039
SHEN Shengyang
118260910042
YAN Shuhan
118260910050

Supervisor:
Chentao WU
Xin XIE

October 29, 2019

1 Introduction

Erasure code is a method of data protection in which data is broken into fragments, expanded and encoded with redundant data pieces and stored across a set of different locations or storage media.

In hadoop system, an erasure code called RS code has already been implemented. For every $\boldsymbol{n}$ data units, RS code will have $\boldsymbol{k}$ parity units, which $\boldsymbol{k}$ is set by user, and generate a matrix which size is $(\boldsymbol{n} + \boldsymbol{k}) \times \boldsymbol{n}$, then store the data that has been transformed by this matrix. Although it will cost more disk space, when any data unit becomes unavailable, we could pick $\boldsymbol{n}$ available units for reconstruction. We can pick these $\boldsymbol{n}$ row in the matrix and calculate its inverse matrix, then multiply the vector to get the origin $\boldsymbol{n}$ data unit.

But RS code has its disadvantage, each time we lost a unit, which is the most probable situation, we have to get all $\boldsymbol{n}$ units for reconstruction, it's a huge cost of time, so maybe we could sacrifice some uncommonly occurred situation, for example lost $\boldsymbol{k}$ units at the same time, to decrease the cost of most situation, for example lost only 1 unit.

LRC code is the solution for such situation, similar to RS code, LRC code also has $\boldsymbol{n}$ data units and $\boldsymbol{k}$ parity units, instead of RS code's $\boldsymbol{k}$ parity units are related to every data unit, which we called global parity unit, LRC code has $\boldsymbol{l}$ local parity units only related to certain data units, and $\boldsymbol{r}$ global parity units where $\boldsymbol{l}+\boldsymbol{r}=\boldsymbol{k}$. In ours project, we choose $\boldsymbol{n} = 6, \boldsymbol{l} = 2, \boldsymbol{r} = 2$. The structure shows in Figure 1.

In the two local parity units, each is related to 3 data units, so that if any data unit is lost, we could use the other two data units and the related local parity unit for reconstruction. That only need half the data reading. Such method can'y handle the situation of four certain units lost, which the units are one local parity unit and the three data units it related to. But this situation is almost impossible to happen so the cost is nearly zero.

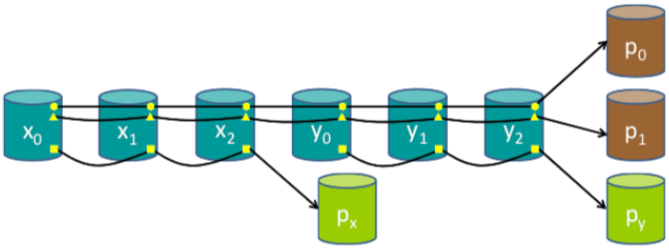


Figure 1: LRC structure

2 Code

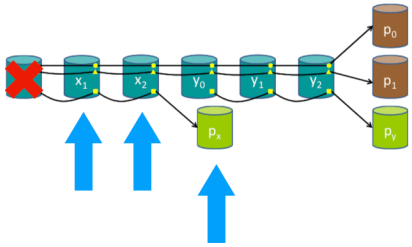
The implementation of LRC code is based on JAVA and composed with following files:

- LRCUtil.java: In this file we define the function to generate the encode matrix and the function to encode data. The encode matrix is composed by an 6×6 identity matrix. Two local parity vectors, one with the first three elements are one and others are zero, another with the last three elements are one and others are zero. Two global parity vectors. All under GF(2^8). The encode matrix shows as below:

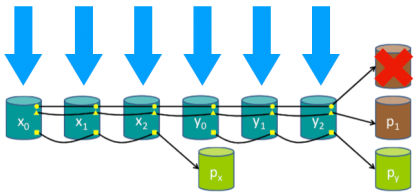
$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 2 & 3 & 4 & 5 & 6 \\ 1 & 4 & 9 & 16 & 25 & 36 \end{bmatrix}$$

- LRCRawErasureCoderFactory.java: In this file we declare the functions we need and merge the options into LRC code.
- LRCRawEncoder.java: In this file we call the functions in LRCUtil.java to generate encode matrix and do the matrix multiplication to encode data.
- LRCRawDecoder.java: In this file we recover the lost data. We separate the matrix into three parts:
  - First local part: this part includes the first three data units and the related local parity unit.
  - Second local part: this part includes the other three data units and the related local parity unit.
  - Global part: this part includes the two global parity units.

  Depend on different situation we will decode under following logic:
  - One unit lost: If this unit belongs to the two local part, we will use the remaining three units to recover the lost unit. If it belongs to the global part, we will use the corresponding row of the encode matrix to re-calculate this unit.

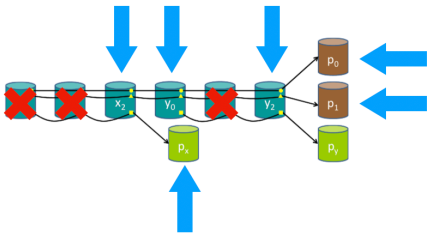


(a) local unit lost



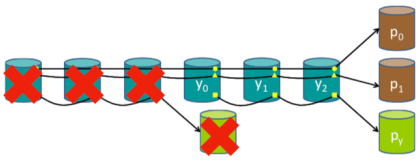
(b) global unit lost

– Two to four units lost: First we get which part does the first unit belong to and use the remaining units in this part, then use the global units if available, if the number of units still lower than 6, we randomly choose units in the last part.



(a) more than one unit lost

– Four units in the same part lost or more than four units lost: This is the situation this LRC code can't recover.



(a) cannot recover

Using the above logic we get the decode matrix either 3×3 or 6×6, then we calculate the invert matrix and recover the lost unit(s).

3 Example

The feasiblity of this LRC code is tested in the hadoop file system in the following steps:

- Compile the whole hadoop project, package it into a native distribution and deploy the hadoop system using docker container.
- Create an empty directory in hadoop file system. Set the erasure code policy to LRC-6-4-1024k.
- Put an 800 MB file in the directory. The file will be encoded by LRC code into 6 data chunks and 4 parity chunks.
- Find the physical locations of file chunks. Choose several chunks and manually delete them.
- Fetch the file from the hadoop file system.
- Verify the identity of the fetched file.

First we compile the whole hadoop project and deploy the hadoop system using docker container. Here our LRC code configuration is composed with 6 data units and 4 parity units, thus we need to deploy 10 datanodes.

Type "hadoop namenode -format" and "$HADOOP_HOME/sbin/start-all.sh" to launch hadoop file system

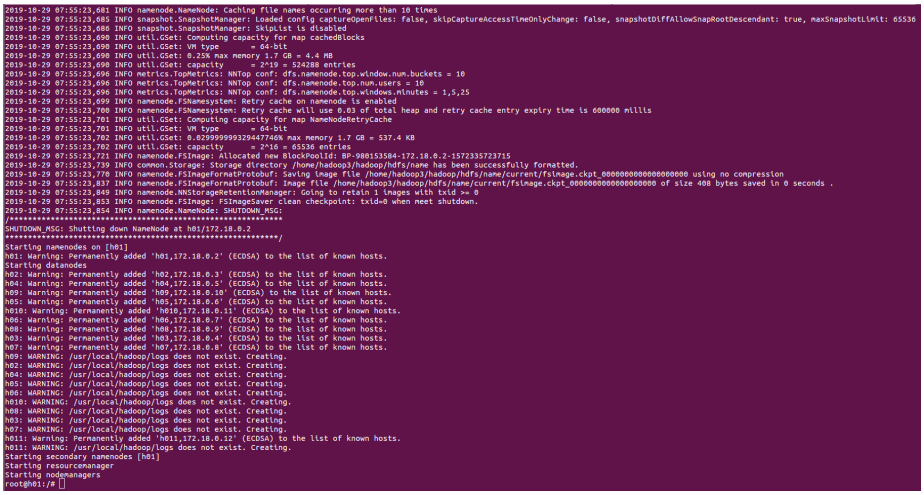


Figure 5: start namenode and datanodes

Type ”hdfs ec -listPolicies” to check if LRC policy is registered in hadoop.

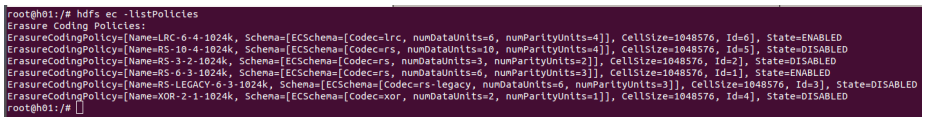


Figure 6: list erasure code policies

Create an empty directory in hadoop file system. Enable the LRC policy and set the erasure code policy to ***LRC-6-4-1024k***.

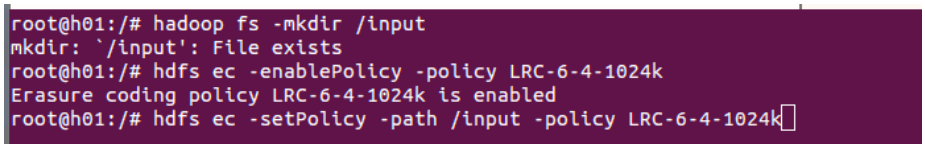


Figure 7: set LRC policy

Type ”hadoop fs -put b.dat /input ” to put an 800 MB file b.dat in the directory /input.

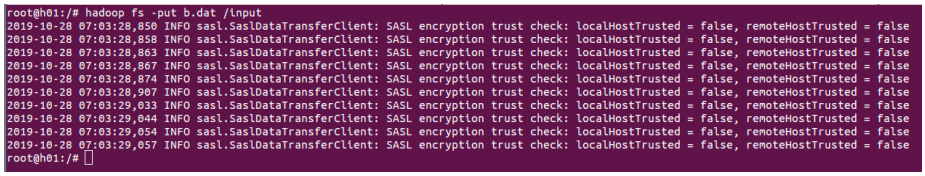


Figure 8: put the file b.dat in /input

Use ”hdfs fsck /input -files -blocks -locations -racks” to find the physical locations of the file in all nodes.

Figure 9: find physical locations

From the above figure the 10 chunks are stored in 172.18.0.11, 172.18.0.6, ..., 172.18.0.4 in order, which respectively correspond to x_0, x_1, x_2, y_0, y_1, y_2, p_x, p_y, p_0, p_1. We delete the data unit x_0.

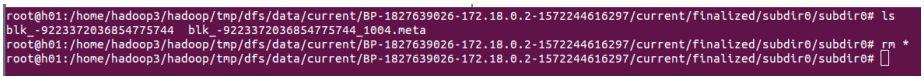


Figure 10: physically delete x_0

Use "hadoop fs -cat /input/b.dat > c.dat" command to fetch the file from hadoop file system.

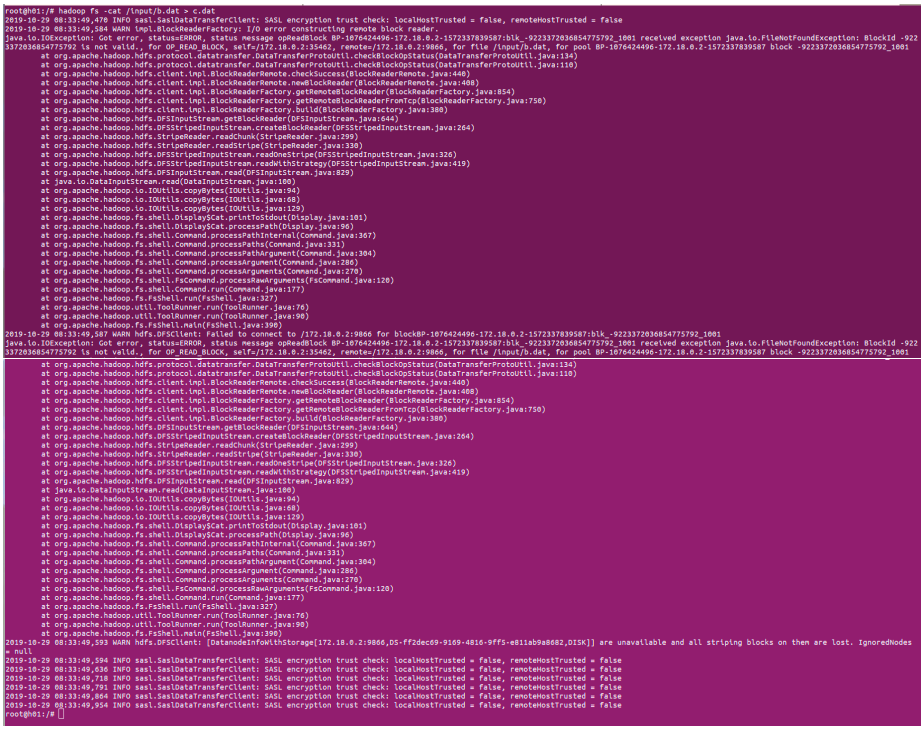


Figure 11: fetch b.dat from hadoop and copy it to c.dat

We can see from the above figure that a block of the target file b.dat is first missing, but then is reconstructed from other blocks. The command outputs correct content to c.dat.

Finally use "cmp" command to verify the identity of c.dat

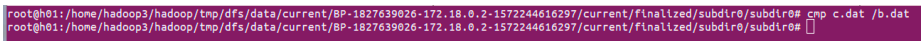


Figure 12: verify the identity of c.dat

We test different missing blocks configurations with successful reconstructions:

- x_0 unit is missing
- x_0 and y_0 units are missing
- x_0, y_0 and y_1 units are missing
- x_0, y_1, p_x and p_0 units are missing

and also an expected failure configuration in which all missing blocks belong to one local part X:

- x_0, x_1, x_2 and p_x units are missing

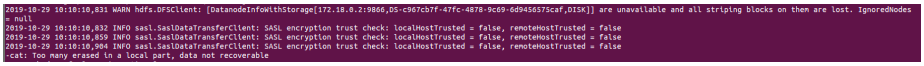


Figure 13: error: too many missing blocks in a local part