分布式式存储系统的纠删码实验－－基于文件系统的模拟实验

一、实验目的

模拟实现分布式系统中的Erasure Code 的实现，采用文件系统来模拟分布式存储系统，编写程序来模拟纠删码的编码和解码过程。

二、实验原理

分布式系统需要在硬件失效等故障发生后仍然能继续提供服务。就数据而言，HDFS采用每份数据3副本的方式，保证某些数据损失之后仍能继续使用。数据的容错除了副本还有另一种做法，就是把丢失的数据计算出来。这就是纠删码的思想了。与多副本对比，纠删码的方式优点在于节省存储空间，缺点在于在存储之前有计算开销，在数据修复的时候也需要计算开销啊。

从数学角度来看，纠删码的原理就是基于线性方程组，例如，本实验中采用的4个数据块，2个校验块，校验块的数据来源于数据块与编码矩阵的乘积，从线性方程的角度来讲，就是4个未知数构成6个线性无关的方程组（编码），任意的4个方程都能够求出4个未知数（解码）。

EC 编码过程：

编码矩阵如下图，Di表示数据块，Ci表示校验块，编码矩阵M组成有两个部分，上面是k*k的单位矩阵，下面是m*k的编码矩阵，本实验采用 Intel EC的范德蒙矩阵。

$$\begin{bmatrix} m_{00} & m_{01} & m_{02} & m_{03} \\ m_{10} & m_{11} & m_{12} & m_{13} \\ m_{20} & m_{21} & m_{22} & m_{23} \\ m_{30} & m_{31} & m_{32} & m_{33} \\ m_{40} & m_{41} & m_{42} & m_{43} \\ m_{50} & m_{51} & m_{52} & m_{53} \end{bmatrix} \cdot \begin{bmatrix} D0 \\ D1 \\ D2 \\ D3 \end{bmatrix} = \begin{bmatrix} C0 \\ C1 \\ C2 \\ C3 \\ C4 \\ C5 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 2 & 4 & 8 \end{bmatrix} \cdot \begin{matrix} D_0 \\ D_1 \\ D_2 \\ D_3 \end{matrix} = \begin{matrix} D_0 \\ D_1 \\ D_2 \\ D_3 \\ C_0 \\ C_1 \end{matrix}$$

校验块：C0 = D0+D1+D2+D3；C1 = D0 + 2*D1 + 4*D2 + 8*D3

EC解码的过程：

以上图的编码矩阵和数据矩阵为例，不妨设此时丢失的数据块石为 D1和C1，EC的解码过程分为两步：

1）根据已有的数据和编码矩阵求出原始数据块。由矩阵的乘法可以看到，在编码过程中，任意的四行已编码数据都是由编码矩阵对应的行数据与原始数据的乘积，所以，显然可知，原始数据可以通过已编码数据与对应编码数据的逆矩阵的乘积来得到。此处D1和C1失效：获取对应的编码矩阵构成解码矩阵；求逆；求原始数据；

$$\begin{bmatrix} D_0 \\ D_2 \\ D_3 \\ C_0 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}^{-1} = \begin{bmatrix} D_0 \\ D_1 \\ D_2 \\ D_3 \end{bmatrix}$$

2）根据原始数据求出所有数据。此时进行的也就是重新编码的过程。

三、实验设计

1. 原始数据：模拟数据流的存储方式，随机生成float 类型的数据，保存于原始数据中，写入本地文件系统，留作备用。

2. 数据存储：简要的模拟分布式系统，采用文件的方式，将分段的数据存储于以数据块号命名的文件中。

3. 数据编码：矩阵运算实现数据编码。

4. 数据失效：采用手动删除文件的方式，模拟文件系统的服务器宕机情况。

四、程序设计

1. 准备阶段：包括原始数据的生成，矩阵运算的准备

2. 数据处理和编码阶段：包括原始数据的分片，编码与存储

3. 数据恢复：原始数据的恢复和数据块的恢复

4. 测试：模拟数据失效，调用数据恢复方法恢复数据

五、程序实现（引用头文件，具体代码片段见附录）

1. 准备阶段(functions.h)

```
//数据初始化，随机生成数据，写入metadata.txt ( prepare.cpp )
void dataInitial();

//矩阵运算 ( matrixOperation.cpp )
void printMatrix(float **m1,int row,int col);
float ** addMatrix(float **m1,float **m2,int row,int col);
float ** multiplyMatrix(float **m1,float **m2,int row,int col1,int col2);
float ** reverseMatirx(float **m1,int row,int col);
void deleteMatrix(float **m1,int row);
```

2.数据处理和编码阶段 （ functions.h ）

```
//数据分片，从文件读取
void dataPartition();
//数据分片预处理，格式化从文件读取的数据
void dataPrepare(List &L,char * buffer);
//数据编码
void erasureCode();
//数据部署
void dataDeploy();
```

3.数据恢复（解码过程）（ functions.h ）

```
//解码调用方法，入口参数：失效数据块号
void recovery(List &tagList);
//获取幸存的数据，构成数据矩阵，并获取解码矩阵M
void serializeData(List &tagList);
//依据数据矩阵和解码矩阵，求出原始数据矩阵，重构metadataMatrix
void decodeToMetaData();
//编码元数据矩阵，构成完整的数据矩阵
void  encodeMetaData();
```

4. 测试 （ simulator.cpp ）

```
void simulator(){
    List tagList;
    char * filename;
    //1.数据块和校验块分别丢失一个数据块
    initList(tagList);
    int dataBlock=0,ecBlock=0;
    addElem(tagList, dataBlock);
    addElem(tagList, ecBlock+BLOCK_NUM);
    //删除数据块对应的文件，模拟失效
    filename = new char[1];
    filename[0]=(char) ((int)'0'+dataBlock);
    remove(filename);
    delete [] filename;
    filename = new char[1];
    filename[0]=(char)((int)'0'+ecBlock+BLOCK_NUM);
    remove(filename);
    delete [] filename;

    recovery(tagList);

    //2.数据块丢失两个数据块
    initList(tagList);
    int * dataBlocks = new int[2];
    dataBlocks[0]=1;
    dataBlocks[1]=2;
    addElem(tagList, dataBlocks[0]);
    addElem(tagList, dataBlocks[1]);
    //删除数据块对应的文件，模拟失效
    for (int i =0; i<2; i++) {
        filename = new char[1];
        filename[0]=(char) ((int)'0'+dataBlocks[i]);
        remove(filename);
        delete [] filename;
    }

    recovery(tagList);
}
```

六、实验结果截图

1. 原始数据的生成和编码：

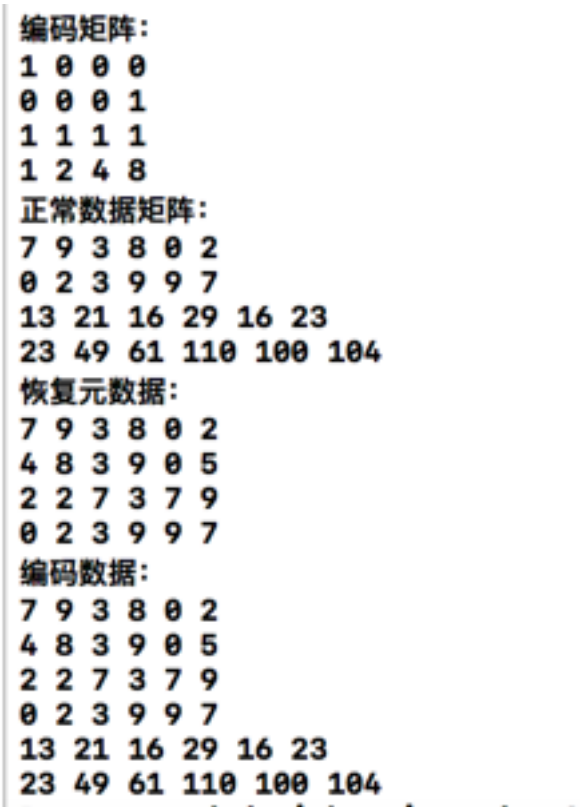
```
元数据已创建
元数据矩阵:
7 9 3 8 0 2
4 8 3 9 0 5
2 2 7 3 7 9
0 2 3 9 9 7
编码数据:
7 9 3 8 0 2
4 8 3 9 0 5
2 2 7 3 7 9
0 2 3 9 9 7
13 21 16 29 16 23
23 49 61 110 100 104
```

2.解码

2.1　　1号数据块和1号校验块失效：

```
编码矩阵:
0 1 0 0
0 0 1 0
0 0 0 1
1 2 4 8
正常数据矩阵:
4 8 3 9 0 5
2 2 7 3 7 9
0 2 3 9 9 7
23 49 61 110 100 104
恢复元数据:
7 9 3 8 0 2
4 8 3 9 0 5
2 2 7 3 7 9
0 2 3 9 9 7
编码数据:
7 9 3 8 0 2
4 8 3 9 0 5
2 2 7 3 7 9
0 2 3 9 9 7
13 21 16 29 16 23
23 49 61 110 100 104
```

2.2　2号和3号数据块失效：



七、实验总结

在实验的实际过程中，遇到了很多问题，其中最大的问题是矩阵的运算问题，特别是矩阵求逆的实现，最开始没有考虑到求逆的具体，所以在实现中数据的声明都是以简单的int整型变量，这时求逆就已经失败了，后来整体更改为float变量，采用LU分解的方式求矩阵逆，但是出了问题，会出现无效数据的情况，所以……后来索性舍去了求逆这一项，直接定义了矩阵的逆矩阵，在解码过程中调用来完成实现……所以总体来讲，实验不是很成功，但是重在理解EC的过程，所以也算成功了。