校园规模的分布式文件共享系统

小组成员

彭昀

张圣明

黄奕桐

胡清泳

杭逸哲

指导老师

邢凯

中国科学技术大学

2018 年 7 月

目录

1. 绪论

1.1 项目背景

1.1.1 分布式文件系统

相对于本机端的文件系统而言,分布式文件系统(Distributed File System， DFS)是一种允许文件通过网络在多台主机上分享的文件系统。在这样的文 件系统中,客户端并非直接访问底层的数据存储区块,而是通过网络,以特定 的通信协议和服务器沟通。借由通信协议的设计,可以让客户端和服务器端 都能根据访问控制清单或是授权,来限制对于文件系统的访问。相对地,在一 个分享的磁盘文件系统中,所有节点对数据存储区块都有相同的访问权,在这 样的系统中,访问权限就必须由客户端程序来控制。

1.1.2 大学生对文件共享的需求

在一所大学中，同学需要频繁的从网络上下载各种文件。这些文件中可能包含音频文件、影音文件，游戏文件；也有可能诸如电子图书、教学课件的文档文件。由于同一校园内同学们所需要的从网络上下载的资源有很大的相同，因此同学们分别到网上去搜索并下载自己需要的资源势必不如从一个校园内的文件系统上下载文件有效率。基于以上几点事实和考量，我们认为可以在校园内搭建一个校园规模的文件共享系统。

1.1.3 万物共享的时代潮流

近年来共享经济发展迅速，共享单车、共享租车、共享充电宝、共享雨伞等莫不如是。诚然，不同用户在实物资源方面的闲置与缺失的矛盾是最突出的，但随着信息互联网基础设施的完善以及人群生活中所接触的必要资源种类日益增多，技术、资源等非实物方面的共享需求将变得更大。例如，很多人的电脑硬盘存储空间大多数时候用不完，过剩也造就了资源的浪费，同时，与之相关的企业数据中心等场所又面临巨大的存储压力。为何不把过剩的存储空间共享出

来？这样不仅资源的利用率将会大大增加，同时存储成本也能大大降低。

1.2 项目概述

1.2.1 项目特点

针对文件共享的传统解决方案主要有资源网站和网盘两种，这两种方案都有其各自的缺陷。搭建资源网站的主要缺点是开销大、可扩展性差、对网络带宽要求较高、隐私性差和存在单点故障的隐患。网盘的缺点则是传输速度慢、受网盘服务提供商约束大，同时私密性无法得到保证。

因此，我们设计的文件共享系统需要具有去中心化、传输速度快、可扩展性高和容错性高的特点。

1.2.2 项目简介

为了实现上述四个特点，我们设想利用校园内同学电脑中的空闲存储空间，将上传的文件分块后存储在网络中的不同电脑中，从而实现存储空间的共享。每个用户既是资源的贡献者，也是资源的享有者、存储者。

2. 项目设计

2.1 架构设计

2.1.1 文件存储

·每个用户贡献出一定的磁盘存储空间，将文件分块后分散存储到各个用户的硬盘上

·纠删码(Erasure Code)对文件进行分块

·DHT 将文件分布存储

2.1.2 文件定位

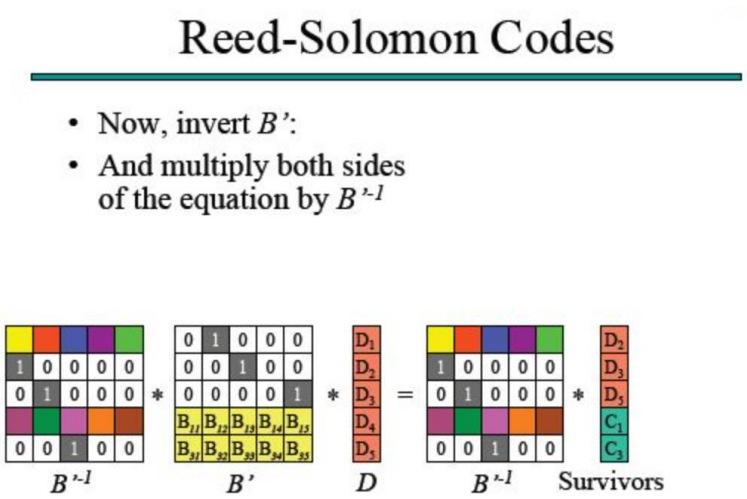
·KAD 路由算法

2.2 理论及技术依据

2.2.1 Erasure Code 的实现

i) Reed-Solomon Codes

RS codes 是基于有限域的一种编码算法，有限域又称为 Galois Field，是以法国著名数学家 Galois 命名的，在 RS codes 中使用 GF(2^w)，其中 2^w >= n + m 。

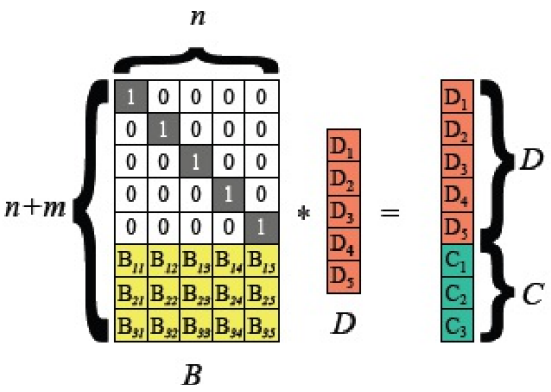


RS codes 定义了一个(n + m) * n 的分发矩阵(Distribution Matrix) （下图表示为 B ）。对每一段的 n 份数据，我们都可以通过 B * D 得到：

假如 D1, D4, C2 失效，那么我们可以同时从矩阵 B 和 B*D 中，去掉相应的行，得到下面的等式：

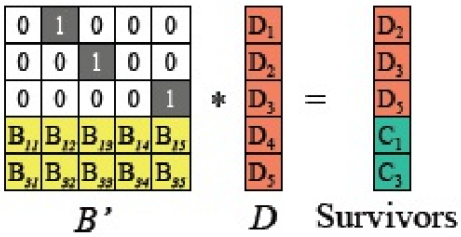
Reed-Solomon Codes

- Codes are based on linear algebra.
  - *B*D* equals an *(n+m)*1* column vector composed of *D* and C (the coding words):



Reed-Solomon Codes

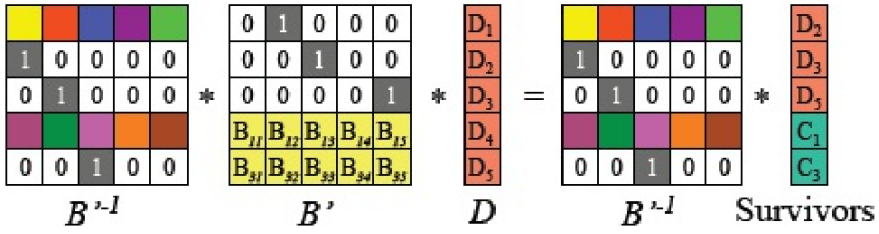
- Suppose *m* nodes fail.
- To decode, we create *B'* by deleting the rows of *B* that correspond to the failed nodes.
- You'll note that *B'*D* equals the survivors.



如果想要从 survivors 求得 D，我们只需在等式的两边左乘上 B'的逆：

Reed-Solomon Codes

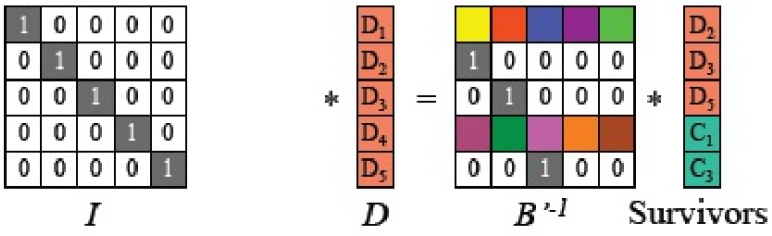
- Now, invert B':
- And multiply both sides of the equation by B'^{-1}



由(B')^-1 * B' = I，所以我们可以计算得到 D：

Reed-Solomon Codes

- Now, invert B':
- And multiply both sides of the equation by B'^{-1}
- Since $B'^{-1} * B' = I$, You have just decoded D!



B 的可逆性可以由柯西（Cauchy）矩阵解决：

柯西矩阵的任意一个子方阵都是奇异矩阵，存在逆矩阵。而且柯西矩阵在迦罗华域上的求逆运算，可以在 O（n^2）的运算复杂度内完成

2.2.2 DHT 网络

a)Kademila 协议简介

Kademlia is a distributed hash table for decentralized peer-to-peer computer networks designed by Petar Maymounkov and David Mazières in 2002. It specifies the structure of the network and the exchange of information through node lookups. Kademlia nodes communicate among themselves using UDP. A virtual or overlay network is formed by the participant nodes. Each node is identified by a number or node ID. The node ID serves not only as identification, but the Kademlia algorithm uses the node ID to locate values (usually file hashes or keywords). In fact, the node ID provides a direct map to file hashes and that node stores information on where to obtain the file or resource.

When searching for some value, the algorithm needs to know the associated key and explores the network in several steps. Each step will find nodes that are closer to the key until the contacted node returns the value or no more closer nodes are found. This is very efficient: Like many other DHTs, Kademlia contacts only O(log(n)) nodes during the search out of a total of n nodes in the system.

Further advantages are found particularly in the decentralized structure, which increases the resistance against a denial-of-service attack. Even if a whole set of nodes is flooded, this will have limited effect on network availability, since the network will recover itself by knitting the network around these "holes".

Kademlia uses a "distance" calculation between two nodes. This distance is computed as the exclusive or (XOR) of the two node IDs, taking the result as an integer number. Keys and Node IDs have the same format and length, so distance can be calculated among them in exactly the same way. The node ID is typically a large random number

that is chosen with the goal of being unique for a particular node (see UUID). It can and does happen that geographically widely separated nodes—from Germany and Australia, for instance—can be "neighbors" if they have chosen similar random node IDs.

Exclusive or was chosen because it acts as a distance function between all the node IDs. Specifically:

1. the distance between a node and itself is zero
2. it is symmetric: the "distances" calculated from A to B and from B to A are the same
3. it follows the triangle inequality: given A, B and C are vertices (points) of a triangle, then the distance from A to B is shorter than (or equal to) the sum of the distance from A to C and the distance from C to B.

These three conditions are enough to ensure that exclusive or captures all of the essential, important features of a "real" distance function, while being cheap and simple to calculate.

b)为何选择 Kademila

Kad 的路由算法天生就支持并发。而很多 DHT 协议（包括 Chord）没有这 种优势。由于公网上的线路具有很大的不确定性（极不稳定），哪怕是同 样两个节点，之间的传输速率也可能时快时慢。由于 Kad 路由请求支持并发，发出请求的节点总是可以获得最快的那个 peer 的响应

实际应用的 DHT 大部分都采用 Kad 及其变种。比如几种知名的 P2P 下载 （BT、eDonkey/电驴、eMule/电骡）的 DHT 都是基于 Kad；知名的 I2P 暗网也依赖 Kad。

2.3 创新点

利用 EC 码进行文件备份，提高了系统存储空间的利用率。

将 DHT 的存储网络和 BT 协议结合起来，这样每一个下载的用户都相当于为 系统中的相应文件进行了一次备份，使得系统的容错性提高。

3. 代码实现

3.1 Erasure Code 模块

3.1.1 模块功能

文件分块

文件校验

文件冗余

3.1.2 模块接口

```
download(char **path,int block_num)
```

对下载回来的文件块进行校验，并恢复成原来的文件

```
upload(char *filename,char *path,char *localblocks_path)
```

对即将上传的文件进行分块、编码并生成 md5 校验码

3.1.3 文件分块

a) 分块规则

提供五种块大小：2K、32K、256K、4M、64M

小于等于 128K 的文件采用 2K 大小的块进行分块

128K~1M 的文件采用 32K 大小的块进行分块

1M~16M 的文件采用 256K 大小的块进行分块

16M~256M 的文件采用 4M 大小的块进行分块

256M 以上的文件采用 64M 大小的块进行分块

块数目一般不超过 64 个，不会少于 4 个。保证既不会因为块数量过多下载困难，也不会因为快数量过少使 erasure code 失去优势。

b) 函数实现

int split(char *filename,char *path,int64_t *blockLen, char **blockname,int64_t *last_blocklen)

filename——原文件名

path—— 原文件所在目录(注意必须以''/''结尾)

blockname—— 分出的数据块的文件名

last_blocklen——最后一个数据块的大小

函数返回值：

成功——返回分得的数据块数

失败——返回对应错误代码

3.1.4 文件校验

a)块文件名规则：

在数据块和编码块均生成后会生成文件块的 16 位 md5 校验值，并写入文件名中，生成的 code 块也会生成 md5 校验值

在恢复文件时，若生成的 md5 校验值与文件名中的不符，则认为文件块失效。

原文件名-最后一块的字节数-数据块数-编码块数-块序号(数据块和编码块单独编号)-1/0(1 代表数据块，2 代表编码块)-md5 校验值.tmp

b)函数实现

```
char *MD5_file (char *path, int md5_len)
```

函数功能：

生成对应文件的 md5 校验值

path——文件路径(包括文件所在目录和文件名)

md5_len——需要生成的 md5 校验值的位数(默认 16 位，可更改)

文件返回值：

成功——返回生成的 md5 校验值的指针

失败——返回对应错误代码

```
int check(char **path, int block_num,int *valid, int datablocknum_int,char
**sequential_path,int64_t *blockLen)
```

函数作用：

检查文件块是否已经损坏，并获取恢复原文件的一些必要信息

path——文件块路径(包括文件块所在目录和文件块名)

block_num——文件块数

valid——指示可用的文件块的数组

datablocknum_int——数据块数

sequential_path——按顺序排列的文件块路径(包括文件块所在目录和文件块名)

blockLen——文件块的长度

文件返回值：

成功——返回 0

失败——返回对应错误代码

3.1.5 文件冗余

a) 说明

采用的是 Erasure Code 中的 LRC，增加了本地块冗余，大大降低 I/O 的消耗(多占用约 5%左右的空间，将 I/O 的消耗降低至没有本地块时的 10~50%。)

集成 GF-complete（伽罗瓦运算库）采用多种矩阵(如柯西矩阵、范德蒙德矩阵)计算编码块，同时优化算法提升计算的速度，尽量降低 erasure code 的计算延迟。

b)函数实现

```
int lrc_init_n(lrc_t *lrc, int n_local, uint8_t *local_arr, int m)
```

lrc——指向 lrc 结构体，其中定义了一些 lrc 的相关参数

n_local——要生成的 local 块的数目

local_arr——每个本地块对应的数据块数

m——全部编码块数(包括 local 编码块和 global 编码块的总数)

函数返回值：

成功——返回 0

失败——返回对应错误代码

函数功能：

初始化 lrc

```
int  lrc_buf_init(lrc_buf_t *lrc_buf, lrc_t *lrc, int64_t chunk_size)
```

lrc_buf——存储数据块和编码块中数据的 buf

lrc——指向 lrc 结构体，其中定义了一些 lrc 的相关参数

chunk_size——每块的大小

函数返回值：

成功——返回 0

失败——返回相应错误代码

函数功能：

初始化 lrc_buf

```
void lrc_destroy(lrc_t *lrc)
```

lrc——指向 lrc 结构体，其中定义了一些 lrc 的相关参数

函数返回值：无

函数功能：

释放分配给 lrc 的内存

```
int lrc_encode(lrc_t *lrc, lrc_buf_t *lrc_buf)
```

lrc——指向 lrc 结构体，其中定义了一些 lrc 的相关参数

lrc_buf——存储数据块和编码块中数据的 buf

函数返回值：

成功——返回 0

失败——返回对应错误代码

函数功能：

对给定的数据块进行编码，生成的编码块中的内容写入 buf

```
int lrc_get_source(lrc_t *lrc, int8_t *erased, int8_t *source)
```

lrc——指向 lrc 结构体，其中定义了一些 lrc 的相关参数

erased——指示哪些块可用的数组

source——指示恢复文件需要那些块的数组

函数返回值：

成功——返回 0

失败——返回对应错误呢代码

函数功能：

判断剩余的文件块能否恢复文件，并给出恢复文件需要的文件块

```
int lrc_decode(lrc_t *lrc, lrc_buf_t *lrc_buf, int8_t *erased)
```

lrc——指向 lrc 结构体，其中定义了一些 lrc 的相关参数

lrc_buf——存储数据块和编码块中数据的 buf

erased——指示哪些块可用的数组

函数返回值：

成功——返回 0

失败——返回对应错误代码

函数功能：

对给定的文件进行译码，将恢复的数据块写入 buf 中

3.1.6 运行流程

a) 编码

通过 upload 接口获得需要分块冗余的原文件

调用 split 函数，检测文件大小，按照文件分块中陈述的规则进行分块

此时分得的块都是数据块，调用 lrc_init_n,lrc_buf_init 函数初始化 lrc 模块

将分得数据块写入 buf 中，调用 lrc_encode 函数生成编码块(4 个 local 编码块+每 4 个数据块 1 个 global 编码块)

所有的数据块和编码块都会调用 md5_file 函数生成 16 位的 md5 校验值

调用 lrc_destroy,lrc_buf_destroy 函数释放申请的内存空间，释放资源

b)译码

通过 download 接口获得需要文件块

调用 check 函数，通过检验文件生成的 md5 值与文件名中的 md5 值来比较，校验文件是否损坏。同时从文件名中获取原文件的一些基本信息，以便恢复原文件

调用 lrc_init_n,lrc_buf_init 函数初始化 lrc 模块

调用 lrc_get_source 函数检测恢复文件需要的文件块以及检验是否能够恢复文件

将 lrc_get_source 函数指定的文件块写入 buf 中，调用 lrc_decode 恢复数据块

调用 combine 函数将数据块合成为原文件

调用 lrc_destroy,lrc_buf_destroy 函数释放申请的内存空间，释放资源

3.1.7 模块测试

模块安装完毕后，利用 upload 接口，编码并分块 test.pdf，得到的结果如下图所示(path 和 localblocks_path 均设为同一路径)：

```
py@py-virtual-machine:~/桌面/lrc-erasure-code-master/test/1$ ls | grep test
test.pdf
test.pdf-39618-24-10-10-0-cd963e86cd7e8787.tmp
test.pdf-39618-24-10-10-1-d4024501bc2658f0.tmp
test.pdf-39618-24-10-1-0-ff61ced9e11f26f5.tmp
test.pdf-39618-24-10-11-1-5a37e2866d592fcb.tmp
test.pdf-39618-24-10-1-1-7854cf77ad1d35b4.tmp
test.pdf-39618-24-10-12-1-950c79fd6c250b43.tmp
test.pdf-39618-24-10-13-1-3bedf9d13d9a9e86.tmp
test.pdf-39618-24-10-14-1-3c44124b27e8ee11.tmp
test.pdf-39618-24-10-15-1-46bd2c50620d2a89.tmp
test.pdf-39618-24-10-16-1-79a835d1a3805a3f.tmp
test.pdf-39618-24-10-17-1-e2cc65ea62e298e9.tmp
test.pdf-39618-24-10-18-1-af29011af65fc406.tmp
test.pdf-39618-24-10-19-1-149447718a71bc6d.tmp
test.pdf-39618-24-10-20-1-de7435a05864c058.tmp
test.pdf-39618-24-10-2-0-3efe85b7b7e694a2.tmp
test.pdf-39618-24-10-21-1-9aaaa8b272aa7754.tmp
test.pdf-39618-24-10-2-1-8835c7293fb90857.tmp
test.pdf-39618-24-10-22-1-0a4c1b0bdf9b4824.tmp
test.pdf-39618-24-10-23-1-fd4f1db658e61ca8.tmp
test.pdf-39618-24-10-24-1-a1f1553c0fc41d39.tmp
test.pdf-39618-24-10-3-0-c9c9dfd0b55d9c18.tmp
test.pdf-39618-24-10-3-1-48a9f4d1a0ac37c1.tmp
test.pdf-39618-24-10-4-0-011a6cfb347987e9.tmp
test.pdf-39618-24-10-4-1-e46232da20183820.tmp
test.pdf-39618-24-10-5-0-547bbf1fc0244f68.tmp
test.pdf-39618-24-10-5-1-54ffef53740b8522.tmp
test.pdf-39618-24-10-6-0-93644fb786f95e8c.tmp
test.pdf-39618-24-10-6-1-03ecf43d5e0ed0c5.tmp
test.pdf-39618-24-10-7-0-8dc8a629bf1bd85f.tmp
test.pdf-39618-24-10-7-1-37162839e8b32803.tmp
test.pdf-39618-24-10-8-0-21fec53c46add02f.tmp
test.pdf-39618-24-10-8-1-e7f8ac5b379ff939.tmp
test.pdf-39618-24-10-9-0-dc94344b203f062d.tmp
test.pdf-39618-24-10-9-1-820c69d0308e9d81.tmp
py@py-virtual-machine:~/桌面/lrc-erasure-code-master/test/1$
```

test.pdf 的 md5 校验值为：

```
py@py-virtual-machine:~/桌面/lrc-erasure-code-master/test/1$ md5sum test.pdf
4752879ed2f90c2a3285f6064d6a1002  test.pdf
```

```
strcpy(path[0],"/home/py/桌面/lrc-erasure-code-master/test/test.pdf-39618-24-10-1-0-ff61ced9e11f26f5.tmp");
strcpy(path[9],"/home/py/桌面/lrc-erasure-code-master/test/test.pdf-39618-24-10-10-1-d4024501bc2658f0.tmp");
strcpy(path[5],"/home/py/桌面/lrc-erasure-code-master/test/test.pdf-39618-24-10-6-1-03ecf43d5e0ed0c5.tmp");
strcpy(path[10],"/home/py/桌面/lrc-erasure-code-master/test/test.pdf-39618-24-10-11-1-5a37e2866d592fcb.tmp");
strcpy(path[25],"/home/py/桌面/lrc-erasure-code-master/test/test.pdf-39618-24-10-1-1-7854cf77ad1d35b4.tmp");
strcpy(path[11],"/home/py/桌面/lrc-erasure-code-master/test/test.pdf-39618-24-10-12-1-950c79fd6c250b43.tmp");
//strcpy(path[12],"/home/py/桌面/lrc-erasure-code-master/test/test.pdf-39618-24-10-13-1-3bedf9d13d9a9e86.tmp");
strcpy(path[12],"/home/py/桌面/lrc-erasure-code-master/test/test.pdf-39618-24-10-6-0-93644fb786f95e8c.tmp");
strcpy(path[13],"/home/py/桌面/lrc-erasure-code-master/test/test.pdf-39618-24-10-14-1-3c44124b27e8ee11.tmp");
strcpy(path[26],"/home/py/桌面/lrc-erasure-code-master/test/test.pdf-39618-24-10-15-1-46bd2c50620d2a89.tmp");
strcpy(path[15],"/home/py/桌面/lrc-erasure-code-master/test/test.pdf-39618-24-10-16-1-79a835d1a3805a3f.tmp");
strcpy(path[16],"/home/py/桌面/lrc-erasure-code-master/test/test.pdf-39618-24-10-17-1-e2cc65ea62e298e9.tmp");
strcpy(path[17],"/home/py/桌面/lrc-erasure-code-master/test/test.pdf-39618-24-10-18-1-af29011af65fc406.tmp");
strcpy(path[18],"/home/py/桌面/lrc-erasure-code-master/test/test.pdf-39618-24-10-19-1-149447718a71bc6d.tmp");
strcpy(path[19],"/home/py/桌面/lrc-erasure-code-master/test/test.pdf-39618-24-10-20-1-de7435a05864c058.tmp");
strcpy(path[14],"/home/py/桌面/lrc-erasure-code-master/test/test.pdf-39618-24-10-2-0-3efe85b7b7e694a2.tmp");
strcpy(path[20],"/home/py/桌面/lrc-erasure-code-master/test/test.pdf-39618-24-10-21-1-9aaaa8b272aa7754.tmp");
strcpy(path[1],"/home/py/桌面/lrc-erasure-code-master/test/test.pdf-39618-24-10-2-1-8835c7293fb90857.tmp");
strcpy(path[21],"/home/py/桌面/lrc-erasure-code-master/test/test.pdf-39618-24-10-22-1-0a4c1b0bdf9b4824.tmp");
//strcpy(path[22],"/home/py/桌面/lrc-erasure-code-master/test/test.pdf-39618-24-10-23-1-fd4f1db658e61ca8.tmp");
strcpy(path[23],"/home/py/桌面/lrc-erasure-code-master/test/test.pdf-39618-24-10-24-1-a1f1553c0fc41d39.tmp");
strcpy(path[27],"/home/py/桌面/lrc-erasure-code-master/test/test.pdf-39618-24-10-3-0-c9c9dfd0b55d9c18.tmp");
strcpy(path[2],"/home/py/桌面/lrc-erasure-code-master/test/test.pdf-39618-24-10-3-1-48a9f4d1a0ac37c1.tmp");
strcpy(path[22],"/home/py/桌面/lrc-erasure-code-master/test/test.pdf-39618-24-10-4-0-011a6cfb347987e9.tmp");
strcpy(path[3],"/home/py/桌面/lrc-erasure-code-master/test/test.pdf-39618-24-10-4-1-e46232da20183820.tmp");
strcpy(path[24],"/home/py/桌面/lrc-erasure-code-master/test/test.pdf-39618-24-10-5-0-547bbf1fc0244f68.tmp");
strcpy(path[6],"/home/py/桌面/lrc-erasure-code-master/test/test.pdf-39618-24-10-7-1-37162839e8b32803.tmp");
strcpy(path[7],"/home/py/桌面/lrc-erasure-code-master/test/test.pdf-39618-24-10-8-1-e7f8ac5b379ff939.tmp");
strcpy(path[8],"/home/py/桌面/lrc-erasure-code-master/test/test.pdf-39618-24-10-9-1-820c69d0308e9d81.tmp");
```

```
py@py-virtual-machine:~/桌面/lrc-erasure-code-master/test$ md5sum test-new.pdf
4752879ed2f90c2a3285f6064d6a1002  test-new.pdf
```

在文件夹中任意取 28 块，进行文件恢复：

校验恢复的文件是否与原文件相同：

可以看到恢复的文件与原文件相同

3.2 DHT 模块

3.2.1 模块组成

dht 网络

文件上传下载网络

守护进程

client 客户端

3.2.2 dht 网络

a) 说明

在 udp 网络上通信，使用了 jech 写的 bt dht 库[13]，对该库进行了一些修改，使得其可以用于

我们的文件系统

b) 通信协议，为 bt dht 协议[14]

ping：ping 节点，看是否存活

find_node：寻找节点

get_peers：寻找拥有块的节点

announce_peer：宣布在当前节点拥有块

c) 函数说明

dht_init：dht 网络的初始化，需要传入非阻塞的 ipv4 或 ipv6 socket

dht_uninit：dht 网络的销毁

dht_ping_node：对节点发送一个 ping 信息，用于添加初始节点

dht_periodic：在每次循环中调用，用于处理各种通信信息

dht_search：发出一个 get_peers 信息，port 号不为 0 时还会发出 annouce_peer 信息，对其实现做了一些修改

dht_dump_tables：存储当前路由表中节点，一般存在.nodes 文件中，对其实现做了一些修改

dht_hash：一个 hash 函数，使用了 MD5

dht_storage_store：在 peers 表中存储对应的文件 hash 信息，从库中提取并修改的一个函数，因为网络中节点需要存储文件

3.2.3 文件上传下载网络

a) 说明

在 tcp 网络里进行，作为一个服务器端对其他节点的文件上传和下载提供服务。由于时间关系，只实现了一个自定义的简单通信协议。

b) 通信协议

upload：上传文件

消息编码：1 bit: ‘u’ + 20 bits: ‘文件名 md5 hash 值’ + 8 bits: ‘文件大小’ + ‘文件’

download：下载文件

消息编码：1 bit: ‘d’ + 20 bits: ‘文件名 md5 hash 值’，返回信息即为文件

c) 函数说明

tcp_recv_store、get_tcp_recv、in_tcp_recv、remove_tcp_recv 以及相应的 send 系函数为 tcp 客户端的存储管理。

tcp_periodic：对各个客户端的通信消息进行处理

3.2.4 守护进程

a) 说明

守护进程运行 dht 网络和文件下载上传网络，并对 client 提供接口。该进程以非阻塞异步 I/O 的方式对各个 socket 进行管理，使用了 select 函数。client 接口以管道的方式提供，守护进程在一个公共管道中监听 client 请求，client 进程创建一个私有管道接收守护进程信息。

守护进程使用 syslog 记录错误信息，用 becomeDaemon 函数变成守护进程。

b) 与 client 通信协议

请求类型：

```
#define DHT_DOWNLOAD 0          // 下载
#define DHT_UPLOAD 1            // 上传
```

回答类型：

```
#define DHT_UNKNOWN_REQUEST 0       // 未知请求
#define DHT_DONE 1                  // 请求完成
#define DHT_TOO_MUCH_REQUEST 2      // 请求过多
#define DHT_MEMORY_NOT_ENOUGH 3     // 内存不足
#define DHT_NODE 4                  // 找到的节点
```

请求：

```
struct server_request {
    pid_t id;
    int type;
    unsigned char info_hash[20];
};
```

向守护进程管道发送以上结构体，其中 id 为进程 id（守护进程由此得知 client 监听的管道），type 为请求类型。

回答：

```
struct server_answer {
    int type;
    struct sockaddr_storage addr;
    int addr_len;
};
```

守护进程向 client 管道发送以上结构体作为回复。

3.2.5 client 客户端

client 客户端是与用户交互的程序，用户运行./client up file 上传文件，运行./client get file 下载文件。程序从用户获取请求后和守护进程通信获得对应节点，再与对应节点的文件下载上传服务器通信完成文件传输。

3.2.6 运行流程

文件上传：client 获得对应文件及文件名 → client 生成文件名 hash → client 与 daemon 进程利用公有管道通信 → daemon 的 dht 网络接到请求，发出 find_node 信息 → dht 网络交互后返回节点信息 → daemon 将节点信息通过 client 管道发送给 client → client 与节点文件服务器

交互上传文件 → 节点文件服务器将接收到的文件 info_hash 存进 peers 表 → 完成

文件下载：client 获得文件名 → client 生成文件名 hash → client 与 daemon 进程利用公有管道通信 → daemon 的 dht 网络接到请求，发出 get_peers 信息 → dht 网络交互后返回节点信息 → daemon 将节点信息通过 client 管道发送给 client → client 与节点文件服务器交互下载文件 → 本地节点 daemon 的 dht 网络发出 announce_peer 信息 → 完成

3.2.7 模块测试（重点信息已用红线画出）

在 tcp 8888，udp 7777 端口运行的 daemon 程序：

```
→  test1 git:(master) ✗ sudo ./dht_daemon
Ping (4)!
Sending pong.
Find node!
Sending closest nodes (0).
  (0+0 nodes.)
Sending find_node for bucket maintenance.
Nodes found (1+0)!
Ping (4)!
Sending pong.
Find node!
Sending closest nodes (0).
  (1+0 nodes.)
Find node!
Sending closest nodes (0).
  (1+0 nodes.)
Sending find_node for neighborhood maintenance.
Nodes found (1+0)!
Find node!
Sending closest nodes (0).
  (2+0 nodes.)
Sending find_node for neighborhood maintenance.
Nodes found (1+0)!
Find node!
Sending closest nodes (0).
  (2+0 nodes.)
Find node!
Sending closest nodes (0).
  (2+0 nodes.)
Sending find_node for neighborhood maintenance.
Nodes found (1+0)!
Find node!
Sending closest nodes (0).
  (2+0 nodes.)
```

```
Find node!
Sending closest nodes (0).
  (2+0 nodes.)
Sending find_node for neighborhood maintenance.
Nodes found (2+0)!
Find node!
Sending closest nodes (3).
  (2+0 nodes.)
Sending find_node for neighborhood maintenance.
Nodes found (1+0)!
Find node!
Sending closest nodes (0).
  (2+0 nodes.)
Get_peers!
Sending nodes for get_peers.
  (2+0 nodes.)
Sending find_node for neighborhood maintenance.
Nodes found (2+0)!
Find node!
Sending closest nodes (0).
  (2+0 nodes.)
Announce peer!
Sending peer announced.
Find node!
Sending closest nodes (0).
  (2+0 nodes.)
Sending find_node for neighborhood maintenance.
Nodes found (2+0)!
Find node!
Sending closest nodes (0).
  (2+0 nodes.)
Find node!
Sending closest nodes (0).
  (2+0 nodes.)
```

在 tcp 8887，udp 7776 端口运行的 daemon 程序：

```
% sudo ./dht_daemon
Sending ping.
Pong!
Sending find_node for neighborhood maintenance.
Nodes found (0+0)!
Find node!
Sending closest nodes (0).
  (1+0 nodes.)
Sending find_node for neighborhood maintenance.
Nodes found (1+0)!
Find node!
Sending closest nodes (0).
  (1+0 nodes.)
Sending find_node for neighborhood maintenance.
Nodes found (2+0)!
Find node!
Sending closest nodes (0).
  (1+0 nodes.)
Find node!
Sending closest nodes (0).
  (1+0 nodes.)
Sending find_node for neighborhood maintenance.
Nodes found (2+0)!
Find node!
Sending closest nodes (0).
  (1+0 nodes.)
Find node!
Sending closest nodes (3).
  (1+0 nodes.)
Find node!
Sending closest nodes (0).
  (1+0 nodes.)
Sending find_node for neighborhood maintenance.
Nodes found (2+0)!
```

```
Sending find_node for neighborhood maintenance.
Nodes found (2+0)!
Find node!
Sending closest nodes (0).
  (1+0 nodes.)
Find node!
Sending closest nodes (3).
  (1+0 nodes.)
Find node!
Sending closest nodes (0).
  (1+0 nodes.)
Sending find_node for neighborhood maintenance.
Nodes found (2+0)!
Get_peers!
Sending found peers.
  (2+0 nodes.)
Find node!
Sending closest nodes (0).
  (2+0 nodes.)
Announce peer!
Sending peer announced.
Sending find_node for neighborhood maintenance.
Nodes found (2+0)!
Find node!
Sending closest nodes (0).
  (2+0 nodes.)
Nodes found (2+0)!
Sending find_node for neighborhood maintenance.
Nodes found (2+0)!
Sending find_node for neighborhood maintenance.
Nodes found (2+0)!
^C%
10:34 josen@joselinux /home/josen/courses/os/X-exciting/dht/test2
% ls
50898f0cb35139d87132f4732a02921300000000  b  dht_daemon  dht_fifo
```

上面标红的文件为上传的文件。

在 tcp 8886，udp 7775 端口运行的 daemon 程序：

```
master# sudo ./dht_daemon
Sending ping.
Pong!
Sending find_node for neighborhood maintenance.
Nodes found (1+0)!
Find node!
Sending closest nodes (0).
  (1+0 nodes.)
Sending find_node for neighborhood maintenance.
Nodes found (2+0)!
Sending find_node for neighborhood maintenance.
Nodes found (1+0)!
Sending find_node for neighborhood maintenance.
Nodes found (2+0)!
Sending find_node for neighborhood maintenance.
Nodes found (2+0)!
Find node!
Sending closest nodes (0).
  (2+0 nodes.)
Sending find_node for neighborhood maintenance.
Nodes found (1+0)!
Sending find_node.
Sending find_node.
Nodes found (2+0)!
Nodes found (1+0)!
Find node!
Sending closest nodes (0).
  (2+0 nodes.)
Sending find_node for neighborhood maintenance.
Nodes found (2+0)!
```

```
Sending find_node.
Nodes found (2+0)!
Nodes found (1+0)!
Find node!
Sending closest nodes (0).
  (2+0 nodes.)
Sending find_node for neighborhood maintenance.
Nodes found (2+0)!
Sending get_peers.
Nodes found (2+0) for get_peers!
Sending get_peers.
Nodes found (2+0) for get_peers!
Got values (1+0)!
Sending find_node for neighborhood maintenance.
Nodes found (2+0)!
Sending announce_peer.
Sending announce_peer.
Got reply to announce_peer.
Got reply to announce_peer.
Sending find_node for neighborhood maintenance.
Nodes found (2+0)!
Find node!
Sending closest nodes (0).
  (2+0 nodes.)
Sending find_node for neighborhood maintenance.
Nodes found (2+0)!
Find node!
Sending closest nodes (0).
  (2+0 nodes.)
Find node!
Sending closest nodes (0).
  (2+0 nodes.)
```

client 服务端，与上面的这一个 daemon 交互。

```
┌──(~/courses/os/X-exciting/dht/test3)
└─(10:30:35 on master ✖ ● ★)──> ls
b  client  client.c  dht_daemon  dht_fifo
┌──(~/courses/os/X-exciting/dht/test3)
└─(10:30:36 on master ✖ ● ★)──> ./client up client.c
┌──(~/courses/os/X-exciting/dht/test3)
└─(10:30:42 on master ✖ ● ★)──> ./client get client.c
┌──(~/courses/os/X-exciting/dht/test3)
└─(10:31:20 on master ✖ ● ★)──> ls
b  client  client.c  client.c2  dht_client_30119  dht_daemon  dht_fifo
┌──(~/courses/os/X-exciting/dht/test3)
└─(10:31:28 on master ✖ ● ★)──> sudo cat client.c2
#include "dht_daemon.h"
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <openssl/md5.h>
#include <sys/stat.h>
#include <sys/wait.h>
#include <fcntl.h>
#include <errno.h>

#define MAX_DOWNLOAD 10

void hash(void *hash_return, int hash_size,
          const void *v, int len)
{
    static MD5_CTX ctx;
    uint8_t digest[16];
    MD5_Init(&ctx);
    MD5_Update(&ctx, v, len);
```

```
        sprintf(fifo_name, CLIENT_FIFO_TEMPLATE, req.id);
        mkfifo(fifo_name, S_IRUSR | S_IWUSR | S_IWGRP);
        write(serverfd, &req, sizeof(struct server_request));
        clientfd = open(fifo_name, O_RDONLY);
        int fd = open(argv[2], O_RDONLY);
        if (fd < 0) {
            perror(NULL);
            exit(EXIT_FAILURE);
        }
        while (1) {
            read(clientfd, &ans, sizeof(struct server_answer));
            if (ans.type == DHT_DONE)
                break;
            if (ans.type == DHT_NODE) {
                upload(&ans.addr, ans.addr_len, info_hash, fd);
            }
            else
                exit(EXIT_FAILURE);
        }
        close(clientfd);
        unlink(fifo_name);
    }
}
┌(~/courses/os/X-exciting/dht/test3)
└(10:31:38 on master ✖ ● ★)──> md5sum client.c2
md5sum: client.c2: 权限不够
┌(~/courses/os/X-exciting/dht/test3)
└(10:31:47 on master ✖ ● ★)──> sudo md5sum client.c2
bb125f7eeac122b99f650af3402751db  client.c2
┌(~/courses/os/X-exciting/dht/test3)
└(10:31:51 on master ✖ ● ★)──> md5sum client.c
bb125f7eeac122b99f650af3402751db  client.c
┌(~/courses/os/X-exciting/dht/test3)
└(10:31:59 on master ✖ ● ★)──>
```

可以看到 client.c 被上传到第二个节点，下载的时候生成的文件是 client.c2，可以看到 md5 值相同。

4. 参考文献

[1] 刘立坤, 武永卫, 徐鹏志,等. CorsairFS:一种面向校园网的分布式文件系统[J]. 西安交通大学学报, 2009, 43(8):43-47.

[2] Stoica I, Morris R, Karger D, et al. Chord: A scalable peer-to-peer lookup service for internet applications[C]// Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications. ACM, 2001:149-160

[3] Lakshman A, Malik P. Cassandra:a decentralized structured storage system[J]. Acm Sigops Operating Systems Review, 2010, 44(2):35-40.

[4] 刘胜利 去 中 心 化 的 安 全 分 布 式 文 件 系 统 上海交通大学计算机科学与工程系 2012

[5] M. Tim Jones (2010-06-04). "Ceph: A Linux petabyte-scale distributed file system" (PDF). IBM. Retrieved 2014-12-03.

[6] "BlueStore". Ceph. Retrieved 2017-09-29.

[7] R. Srikant and Dongyu Qiu. Modeling and performance analysis of BitTorrent-like peer-to-peer networks

[8] Kaashoek M F, Karger D R. Koorde: A Simple Degree-Optimal Distributed Hash Table[J]. 2003, 2735:98-107.

[9]Zhang H, Wen Y, Xie H, et al. Distributed Hash Table[M]. Springer New York, 2013.

[10] Boyer E B, Broomfield M C, Perrotti T A. GlusterFS One Storage Server to Rule Them All[J]. 2012.

[11] Donovan S, Symposium L, Kleen A, et al. Lustre: Building a File System for 1,000-node Clusters[J]. Proceedings of the Linux Symposium, 2003:9.

[12] https://github.com/baishancloud/lrc-erasure-code

[13] https://github.com/jech/dht

[14] http://www.bittorrent.org/beps/bep_0005.html