国防科技大学

分布式系统项目报告

（研究生）

项目题目（中文）	分布式文件系统的设计与实现
项目题目（外文）	**Design and Implementation of a Distributed File System**
学 生 姓 名	蔡孟栾
学生所属学院	计算机学院
学 号	**20023131**
年 级	**2020 级**

目录

小组成员

姓名	蔡孟栾	邬小军
学院	计算机学院	计算机学院
学号	20023131	20023091

项目环境

操作系统：Windows 10 64 位

编程语言：Python 3.8

编译器：JetBrains PyCharm

配置环境：使用到的依赖包见代码目录中的 requirements.txt 文件

一、选题目的

1. 巩固对分布系统式文件服务系统的基本功能和原理的认识。
2. 尝试学习大型复杂系统中负载均衡和容灾备份，并在本系统中实践运用。
3. 加深对分布式系统课程所学知识的理解，锻炼开发能力。

二、设计原理

2.1 相关工作

本项目参考了 TFS 开源项目，本节主要从分布式文件系统的简要概述、TFS 的架构、设计初衷和对本系统设计与实现的启发等展开描述。

2.1.1 分布式文件系统概述

分布式系统的主要目标是实现共享资源，共享存储信息可能是分布式资源共

享的一个最重要的方面。分布式存储也主要用来解决以下问题：海量数据存储问题、数据高可用问题（冗余备份）问题、较高的读写性能和负载均衡问题、支持多平台多语言问题、高并发问题等。

随着数据量越来越多，在一个操作系统管辖的范围存不下了，那么就分配到更多的操作系统管理的磁盘中，但是不方便管理和维护，因此迫切需要一种系统来管理多台机器上的文件，这就是分布式文件管理系统 。分布式文件系统的文件服务允许用户在企业内部网上的任一计算机上访问自己的文件，程序可以像对待本地文件一样存储和访问远程文件。

TFS、HDFS 等都不是系统级的分布式文件系统，而是应用级的分布式文件存储服务。所以他们的不同的结构、功能设计，是用以解决不同的实际需要。
大数据存储解决方案需要满足高可扩展性,分布式文件系统因其良好的可扩展性成为大数据存储系统的核心。元数据是新型分布式文件系统数据访问模型的关键组成部分。GFS、TFS 等利用这种架构的分布式文件系统存储数据。

2.1.2 TFS 的设计目标

TFS 的设计初衷是为了解决淘宝网站海量小文件的存储问题。TFS 文件存储系统是一个分布式的文件系统，分为两部分 NameSever 节点和 DataServer 节点，可以运行在同一台服务器上，但是 TFS 一般搭建集群，由两个 Nameserver 节点和若干个 DataServer 节点组成，两个 NameServer 节点分别为一主一备,提高系统的安全性。TFS 文件存储系统结构如下图所示[1]。TFS 文件存储结构也采用了块的概念，TFS 文件存储系统中的块的大小默认 64 M，但是可以根据需求更改配置项更改块的大小。NameServer 负责 DataServer 的状态管理，并维护块与 DataServer 的映射关系。NameServer 不负责实际数据的读写，实际数据的读写由 DataServer 完成。

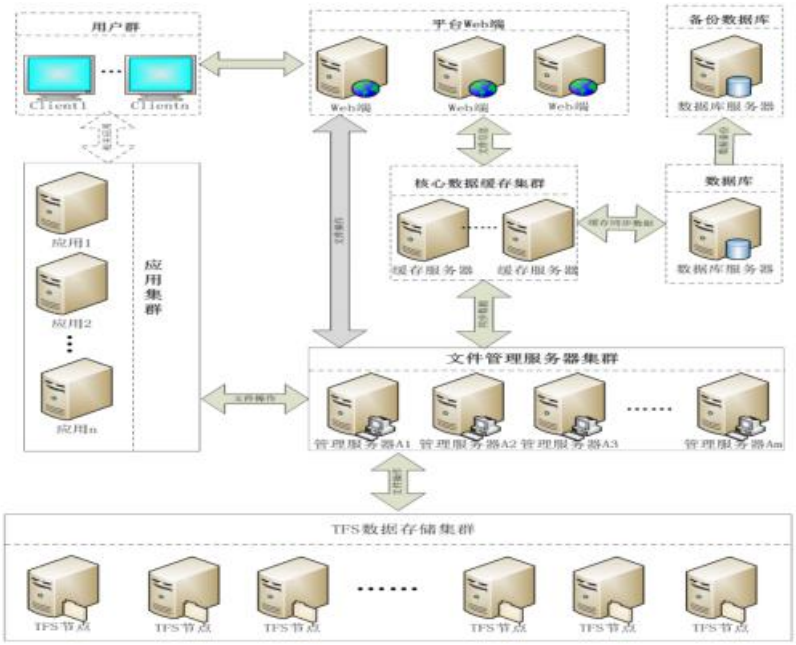

图 1 TFS 文件存储系统结构图

2.1.3 TFS 的原理架构

TFS 是一个高可扩展、高可用、高性能、面向互联网服务的分布式文件系统，主要针对海量的非结构化数据。因为淘宝对小文件存储的需求特性，TFS 设计也就为淘宝提供海量的小文件存储，文件大小不超过 1M，TFS 也被广泛应用在淘宝的各项应用中。TFS 的架构图如下所示。

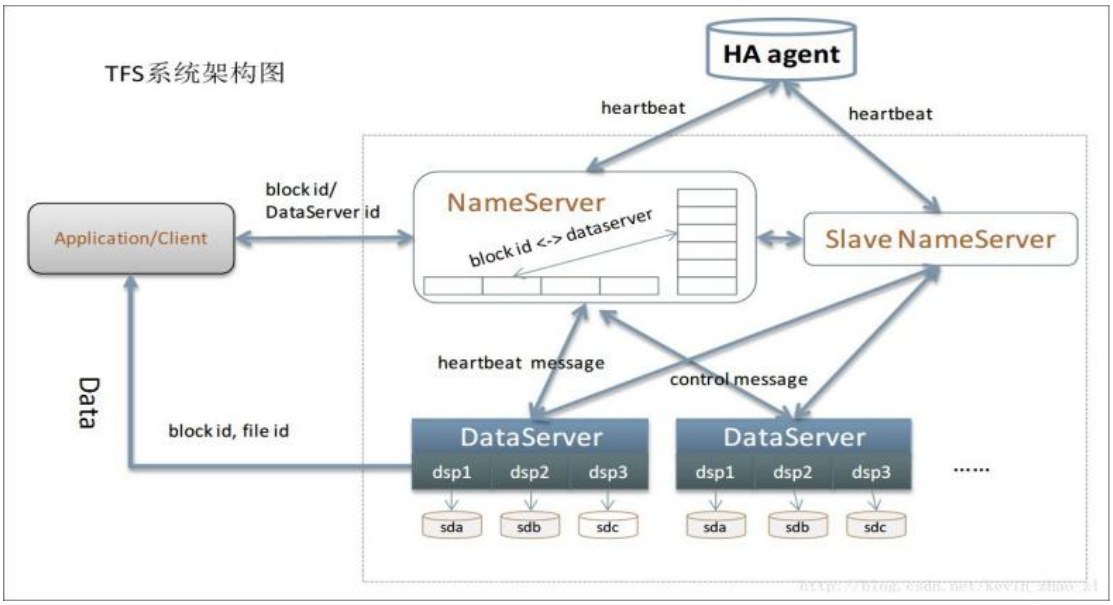

图 2 TFS 系统架构图

其中包含的主要结构有：

Master Nameserver：主目录服务器，用于管理 dataserver 集群信息，缓存 dataserver 中的 block 信息。（监控所有数据节点的运行状况，负责读写调度的负载均衡，同时管理一级元数据用来帮助客户端定位需要访问的数据节点）

Slave Nameserver：从目录服务器，功能和 Master Nameserver 一致，解决了 Master Nameserver 的单点失效问题。

DataServer：多个独立节点，用于存储文件信息，Dataserver 是以 block 的形式进行存储，一个 block 会在多个 dataserver 进行备份，当 master block 所属的 dataserver 宕机了，依旧可以用其他 dataserver 的 block。（负责数据实际发生的负载均衡和数据冗余，同时管理二级元数据帮助客户端获取真实的业务数据。）

Client：客户端对分布式文件系统进行访问和修改。

HA agent：高可用工作机制代理，主要是实现 Master Nameserver 和 Slave Nameserver 的快速切换。

Heartbeat：心跳机制，对各个节点的状态加以检测。

在客户端向该文件系统发出请求访问时，首先会经过 Nameserver，Nameserver 会定位到具体文件的机器信息、块信息、文件信息给 client 端，然后 client 端再访问对应的 dataserver 服务器对文件进行操作。

TFS 部署了 Master Nameserver 和 Slave Nameserver ，并且部署了 HA agent 来负载主、从服务器的切换，主、从服务器都要定时地向 HA agent 发送心跳检测，来告诉 HA agent“我还活着”。一旦接收不到服务器的心跳请求，HA agent 会快速地进行过主机切换，保证系统正常运行。同时，DataServer 也要定时地向 Master Nameserver 发送心跳请求。

2.1.4 对本项目的启发

本系统服务器端采用的是目录服务器和节点服务器两层架构（这部分会在 2.2 节介绍），系统的核心就在于 Directory server（主目录服务器），它主要为了实现文件上传下载转发、节点负载均衡、版本控制、锁机制、节点控制等功能。由于目录服务器是整个系统的控制中心，而且可能会存在单点失效的情况。

面对大流量的外界访问，如果没有限流手段，主目录服务器很可能会成为整个系统的性能瓶颈所在。一旦主目录服务器宕机，整个分布式系统就会彻底瘫痪。

因此增加系统的容错手段来提升系统的高可用是极其重要的。文件系统采用Master-Slave（主/从模式）是容错和提高系统高可用的很重要的手段。为了防止主/从服务器出现宕机的情况，需要设置主、从模式，在一个时间段里只能有个主，如运行期间主服务器出现问题（如：挂死、网络失联）都会进行切主逻辑，以便让整个服务能正常运行。

不管是怎样的分布式系统，其主要的操作都是读写信息,这便容易产生数据一致性问题,在这里我们假设分布式文件系统的服务器集群中的每一个节点都可进行读写的话，则可能出现数据同步混乱。（分布式文件系统的并发性和缺乏全局的统一时间的特性导致的,所以可能在任意时刻中各个节点上都发生了数据更新，我们无法判断哪个数据的更新操作是先执行还是后执行，也就是无法保证更新的顺序性，同时因为节点更新后数据同步是异步的，所以便可能产生了因为同步延迟的问题导致最新的数据被旧数据覆盖掉）

在分布式系统中也存在网络时延的问题，假如有两个节点都要对数据 A 进行修改操作，可能会产生数据不一致性错误，同时在同步上因为多个节点都可以自由同步，则存在拜占庭将军问题，而且会增加开发复杂度大大，并且在出现问题的时候，很难寻找问题。

而采用主/从模式的话，专门用一个节点来充当主节点便能够确保事务执行的顺序性，防止新数据被老数据覆盖的情况。同时，一旦主服务器发生宕机，可以使用备用服务器来保证系统正常运行。

TFS 采用了 Master-Slave 主/从模式来处理上述问题。我们学习了 TFS 等的 Master-Slave 模式，借鉴了其中的部分思路，新增了备用目录服务器，对主目录服务器进行了整体备份。备用目录服务器在正常情况是不执行操作的，一旦客户端向主目录服务器发送请求长时间没有响应的话,会重新向备用目录服务器发送请求，备用目录服务器会代替执行相关操作。这样即使在主目录服务器发生宕机时，备用目录服务器也能接管客户端的访问请求继续工作，借以提高整个系统的可用性和稳定性。

2.2 系统整体分析

2.2.1 设计原则

我们学习了 TFS 等开源项目的一些优点，结合分布式系统的特点，将保持文件一致性作为实现的原则，旨在实现一些功能性的需求，保持系统的稳定性。而一些具体的读写性能作为次要因素考虑，只实现了简单的负载均衡、容灾备份机制，没有针对文件的存取做特定优化。

2.2.2 系统架构

本项目采用 Client-Server(C/S)结构，即客户端-服务器模型，服务器负责数据的管理，客户机负责完成与用户的交互任务。与 C/S 结构通常采取的两层结构不同，本系统服务器采用两层结构，即节点服务器和目录服务器。节点服务器主要用于文件的存储和备份，与客户端直接进行文件下载、上传等操作；目录服务器主要用于收集节点服务器群的文件信息，以响应客户端的各类请求。系统整体结构如下图所示。

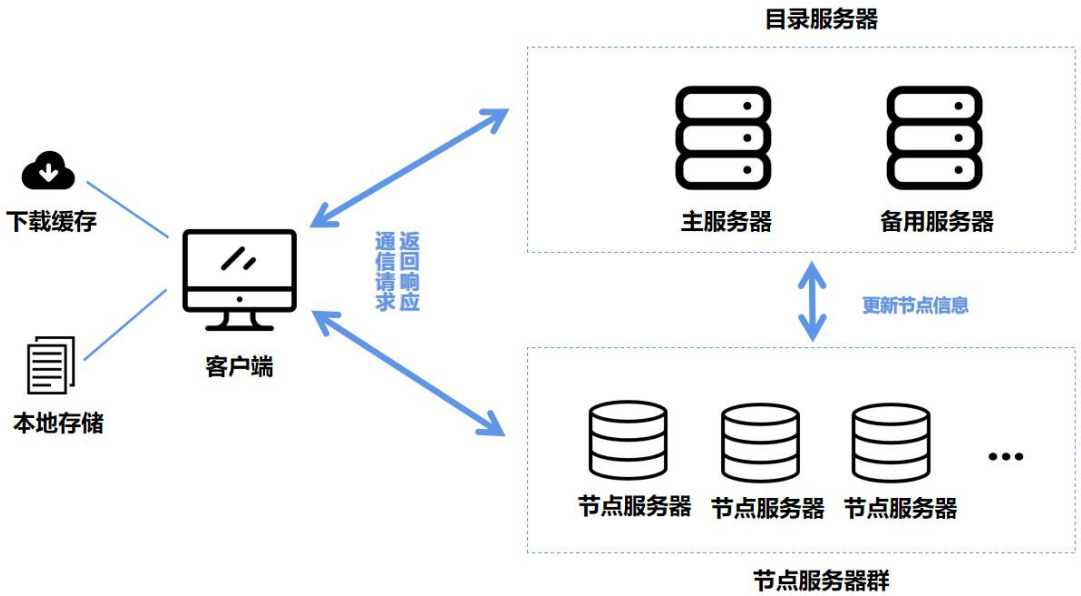

图 3 系统整体结构图

目录服务器包括主目录服务器和备用目录服务器，每个目录服务器都有一个系统备份的文件夹，和目录服务器表。最核心的部分是目录服务器表的结构，因为客户端每次的交互都需先通过目录服务器。目录服务器表由节点文件信息、连

接节点信息、文件读写锁信息和节点权重信息组成，具体结构如下图所示。

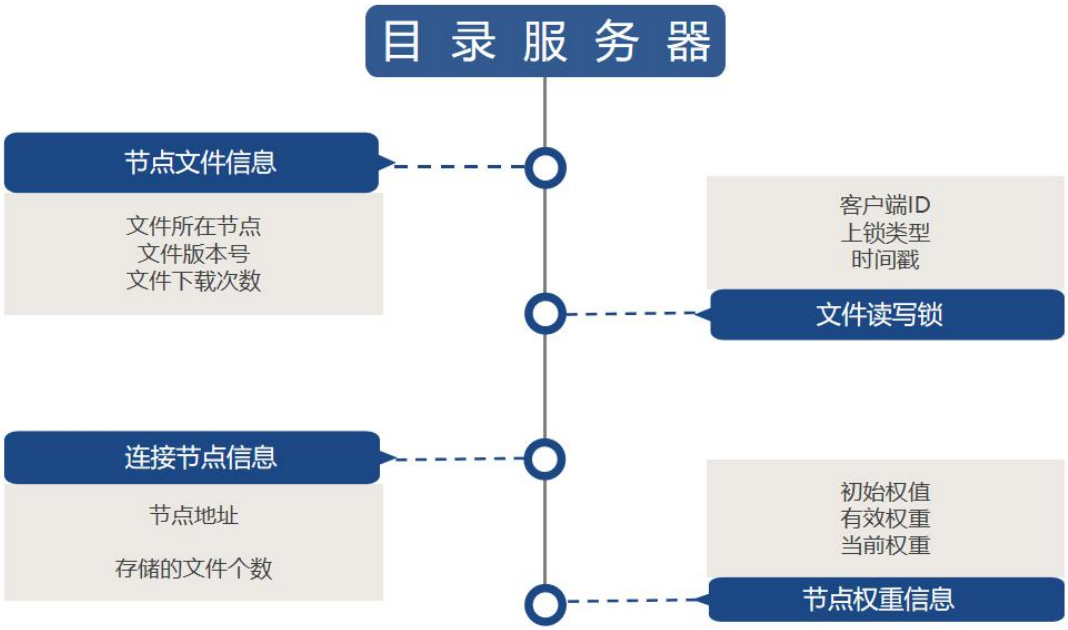

图 4 目录服务器表结构图

节点服务器之间互相独立，各自存储不同的文件信息，同时节点之间也可以互相通信。针对节点服务器文件的管理，我们采用了负载均衡的算法尽量保证每个节点服务器存储文件的数量大致相同，防止某单一节点服务器压力过大而崩溃。由于节点服务器之间的独立性，那么对文件系统来说，新增或删除节点服务器就变得容易，文件系统的可扩展性强。

客户端我们引入了 Cashing File，主要是在完成客户端的开发时，目光更多聚焦在如何让本地的读写效率更高，便采用了缓存的方式以提高客户端的读写效率。

2.3 项目代码结构

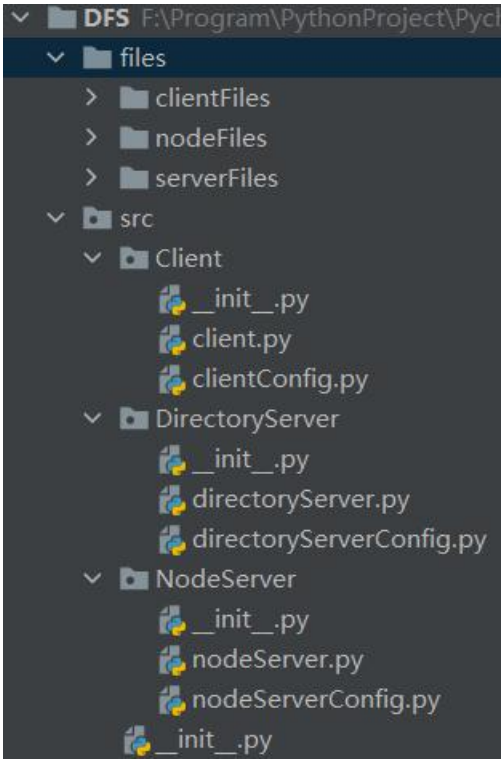

图 5 项目代码结构图

上图为本项目的代码结构。分为两个目录。下文是每个文件的简要说明。（__init__.py 为 Python 包的初始化文件）

1. files —— 文件存放目录
    - clientFiles —— 存放客户端文件
    - nodeFiles —— 存放节点服务器文件
    - serverFiles —— 存放目录服务器文件
2. src —— 源代码目录
    - Client 包
        - client.py —— 客户端功能实现
        - clientConfig.py —— 客户端配置文件
    - DirectoryServer 包
        - directoryServer.py —— 目录服务器功能实现
        - directoryServerConfig.py —— 目录服务器配置文件
    - NodeServer 包

- nodeServer.py —— 节点服务器功能实现
- nodeServerConfig.py —— 节点服务器配置文件

三、功能实现

本系统实现的功能主要有文件的上传、下载、查询和删除，缓存机制、负载均衡、备份机制、崩溃处理、读写锁及版本控制等。

3.1 客户端主界面

因考虑到分布式文件服务系统更适合作为 PC 端软件提供给用户使用，所以界面设计没有采用网页的形式，而是采用 GUI 客户端的形式。GUI 编程采用 Python 的 PySimpleGUI 模块实现。界面如下图所示。

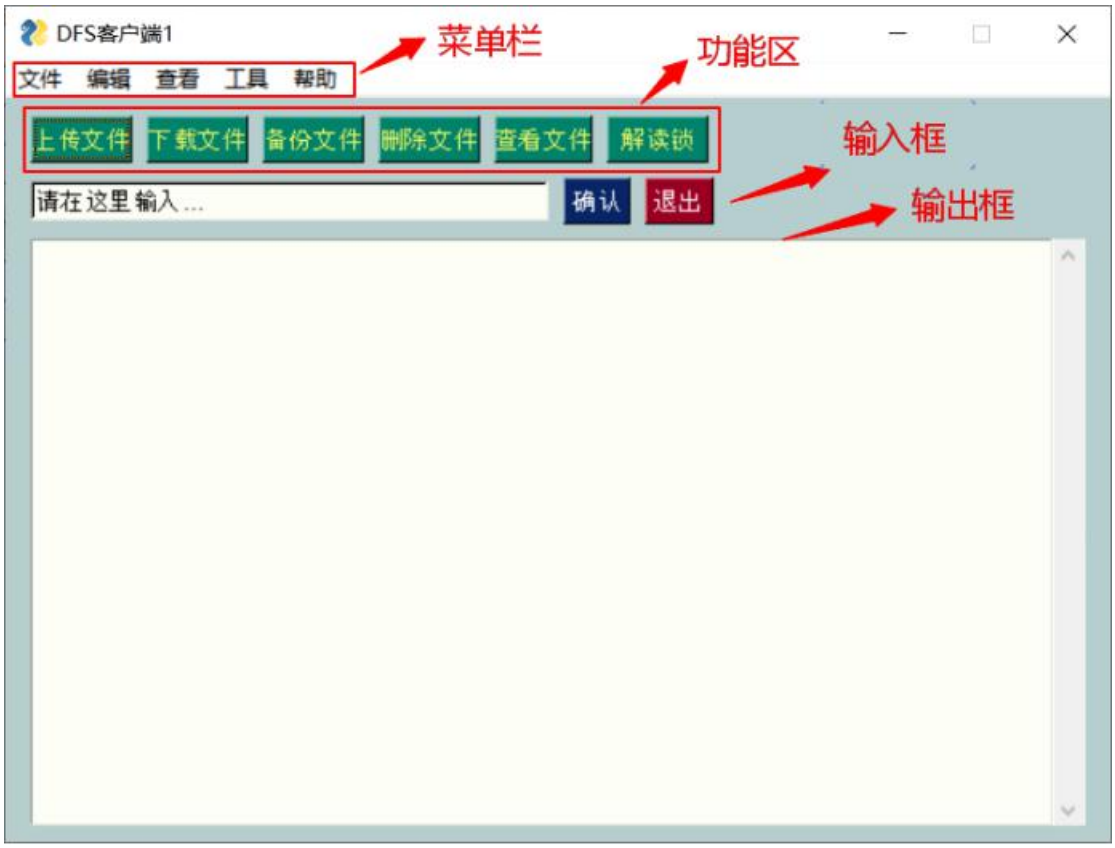

图 6 用户界面图

界面最上面一栏为菜单栏，每个菜单都有相应的下拉选项，这里不做过多阐

述。在菜单栏下面有一排绿色的按钮为功能区，点击即可选择相应功能。接下来是输入框和输出框，分别用于输入信息和输出信息。

3.2 文件上传模块

文件上传模块主要功能是将客户端的本地文件上传到服务器端，并对上传的文件进行备份，对重复上传的文件设置了合理机制，避免无必要的重复上传而导致资源浪费，同时实现引入了负载均衡机制，实现了文件上传时的均衡性，使得文件上传过程资源分配更合理、资源利用更有效。

3.2.1 模块设计

客户端将文件上传到系统时，首先需保证文件置于本地的客户端文件夹中，才能进行上传。输入上传的文件名，目录服务器端会根据上传的负载均衡算法，自动算出要上传的节点服务器 ID，然后将文件上传到该节点服务器上，同时文件也将会备份到系统备份文件夹中，更新文件的版本号以及目录表的信息。文件上传的交互原理图如下所示。

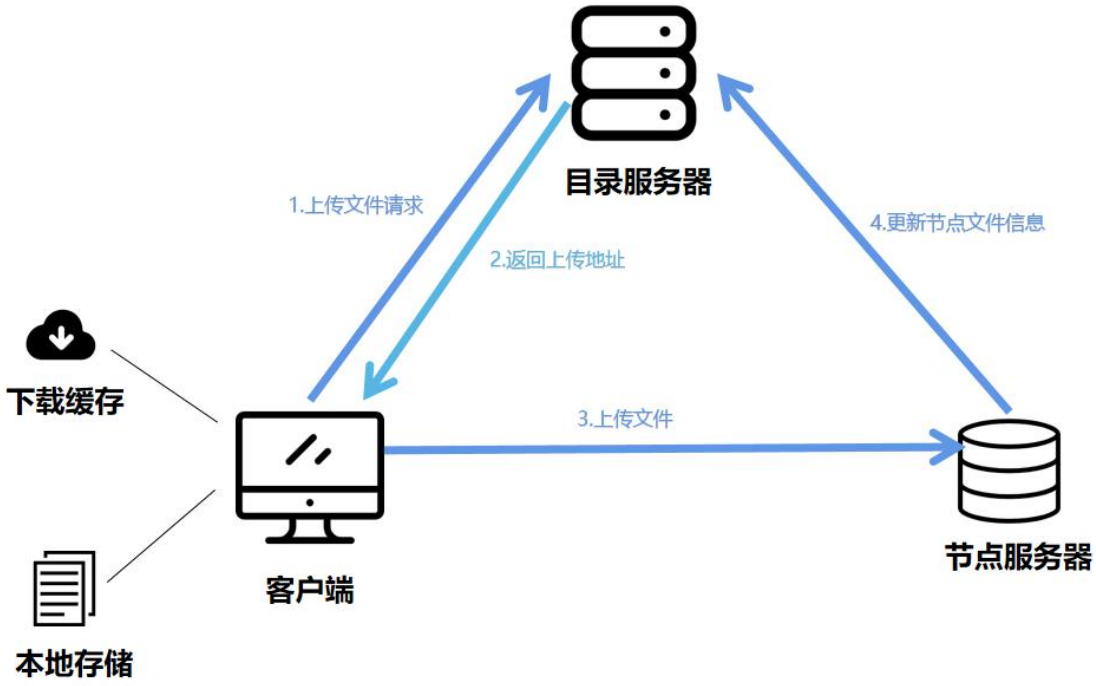

图 7 文件上传

若客户端再次上传同名文件，即二次上传时，目录服务器端会首先询问用户是否要覆盖原文件，如果选择覆盖原文件，那么该过程和上传一个新的文件本质一样；如果选择不覆盖原文件，那么服务器端会进一步询问用户是否创建一个新文件，如果用户选择创建一个新文件，那么在不影响原文件的基础上会新建一个新文件，新文件即为原文件的副本，如果用户选择不创建一个新文件，那么服务器会视为用户放弃对文件的修改，所以不会对文件做任何更改。

3.2.2 用例演示

这是文件二次上传的测试用例，如上节介绍，我们选择不覆盖原文件并新建一个文件，则客户端本地和节点服务器都会产生一个文件名为原文件名(1)的文件。具体效果如下图所示。

图 8 文件上传用例演示结果

3.3 文件下载模块

文件下载模块主要功能是将服务器端的文件下载到客户端本地，并对下载的

文件添加缓存，对重复下载的文件设置了合理机制，避免无必要的重复下载而导致资源浪费，同时实现引入了负载均衡机制，实现了文件下载时的均衡性，使得文件下载过程资源分配更合理、资源利用更有效。

3.3.1 模块设计

客户端要从文件系统中下载文件时，首先可以查看到当前系统中已存在的每个文件，以及其存放的节点位置，然后用户决定是上读锁还是上写锁，输入文件名进行下载，如果下载成功，将下载到客户端本地的文件夹以及客户端的缓存文件夹，客户端中的缓存文件表也会更新，缓存的文件表中带上了此时下载的时间戳以及文件的版本号。文件下载的交互原理图如下所示。

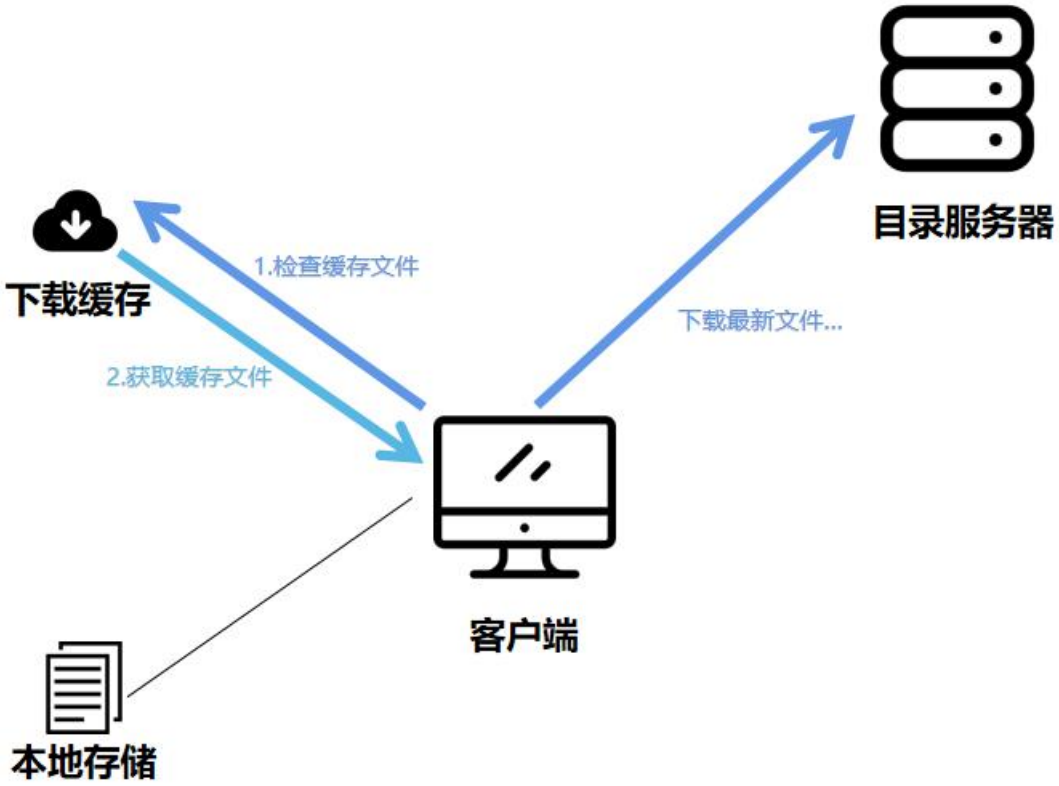

图 9 文件下载

文件成功下载到本地后，客户端会自动备份到缓存文件夹中，缓存文件夹中的内容会存在 10min，若用户在此时间内再次下载该文件，系统会让你选择是否在缓存文件夹中下载，选择“是”，则在缓存文件夹中下载，选择“否”，则在相应的节点服务器（加权轮巡）中下载。

3.3.2 用例演示

这是文件下载的测试用例，如上节介绍，我们下载一个文件后再次下载该文件，并选择从缓存文件夹中下载。具体效果如下图所示。

图 10 文件下载用例演示结果

3.4 文件查询删除模块

文件查询模块主要功能是查询客户端本地文件、查询服务器文件以及查询备份文件。文件删除模块主要功能是将所有节点上的该文件及系统备份文件都进行删除。

3.4.1 模块设计

文件查询时，用户可选择查询客户端本地文件或服务器文件或备份文件夹文件。查询服务器文件时会显示文件所在节点及文件数。文件查询的交互原理图如下所示。

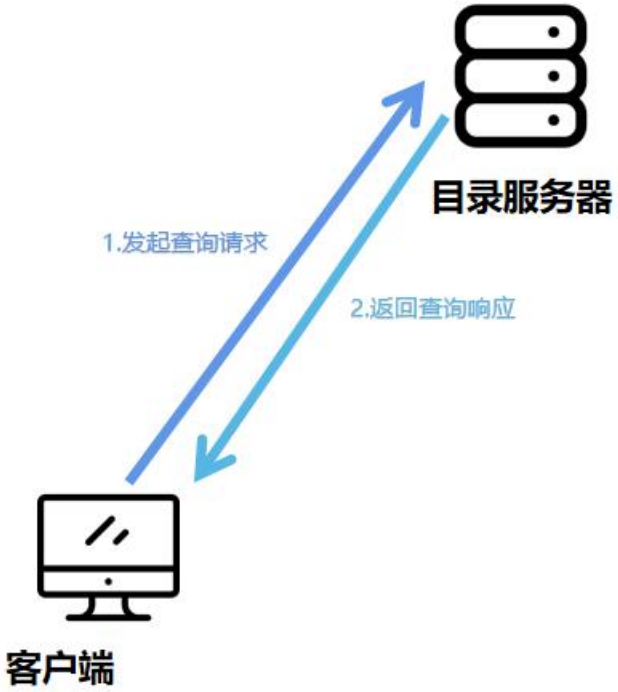

图 11 文件查询

删除文件时，输入要删除的文件名，删除成功后系统会将所有节点上的该文件及系统备份文件都进行删除。文件删除的交互原理图如下所示。

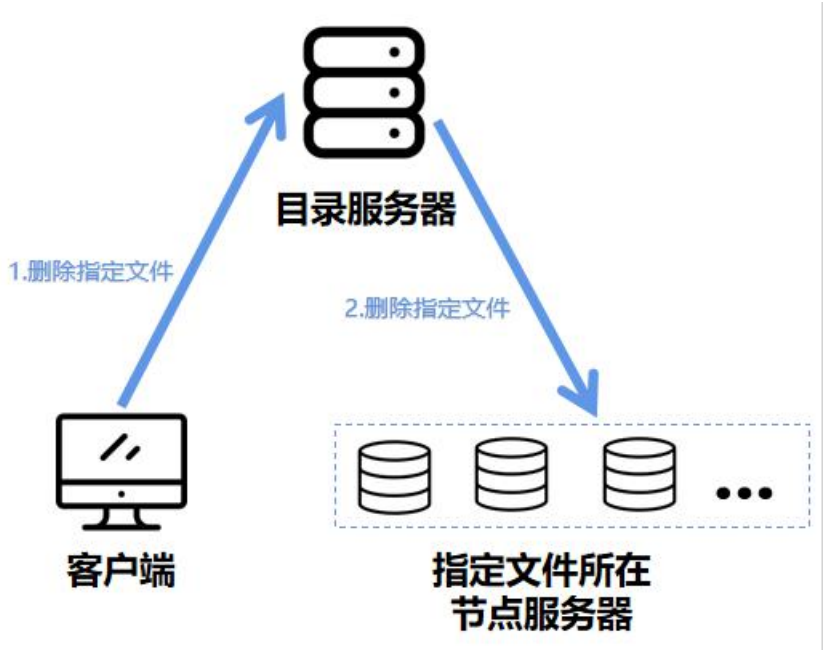

图 12 文件删除

3.4.2 用例演示

这是文件查询的测试用例，这个功能相对简单不用做过多介绍，具体效果如下图所示。

查询本地文件

```
Files stored in client folder.
------------------------------
222.txt
3.txt
```

查询服务器文件

```
Files stored on the server
------------------------------
1.txt,  所在的节点：['http://127.0.0.1:5001/', 'http://127.0.0.1:5002/', 'http://127.0.0.1:5004/', 'http://127.0.0.1:5003/'],  节点总数：4
2.txt,  所在的节点：['http://127.0.0.1:5002/'],  节点总数：1
22.txt,  所在的节点：['http://127.0.0.1:5004/', 'http://127.0.0.1:5002/', 'http://127.0.0.1:5003/'],  节点总数：3
3.txt,  所在的节点：['http://127.0.0.1:5003/'],  节点总数：1
333.txt,  所在的节点：['http://127.0.0.1:5004/', 'http://127.0.0.1:5003/'],  节点总数：2
4.txt,  所在的节点：['http://127.0.0.1:5004/'],  节点总数：1
4444.txt,  所在的节点：['http://127.0.0.1:5004/'],  节点总数：1
```

查询备份文件

```
Files all backup on the server
------------------------------
1.txt,  所在的服务器：http://127.0.0.1:5000
111.txt,  所在的服务器：http://127.0.0.1:5000
2.txt,  所在的服务器：http://127.0.0.1:5000
22.txt,  所在的服务器：http://127.0.0.1:5000
222.txt,  所在的服务器：http://127.0.0.1:5000
3.txt,  所在的服务器：http://127.0.0.1:5000
333.txt,  所在的服务器：http://127.0.0.1:5000
4.txt,  所在的服务器：http://127.0.0.1:5000
4444.txt,  所在的服务器：http://127.0.0.1:5000
5.txt,  所在的服务器：http://127.0.0.1:5000
download_test.txt,  所在的服务器：http://127.0.0.1:5000
upload_test(1).txt,  所在的服务器：http://127.0.0.1:5000
upload_test.txt,  所在的服务器：http://127.0.0.1:5000
```

图 13 文件查询用例演示结果

这是文件删除的测试用例，这个功能相对简单不用做过多介绍，具体效果如下图所示。

图 14 文件删除用例演示结果

3.5 负载均衡

负载均衡，英文名称为 Load Balance，其含义就是指将负载（工作任务）进行平衡、分摊到多个操作单元上进行运行，例如 FTP 服务器、Web 服务器、企业核心应用服务器和其它主要任务服务器等，从而协同完成工作任务。

负载均衡构建在原有网络结构之上，它提供了一种透明且廉价有效的方法扩展服务器和网络设备的带宽、加强网络数据处理能力、增加吞吐量、提高网络的可用性和灵活性。

本系统针对文件的上传和下载两个模块设计负载均衡机制，实现了节点服务器存储文件的均衡性，避免某个节点压力过大而导致效率降低甚至崩溃，这样使得资源分配更合理、资源利用更有效。本节分为两个部分展开，分别介绍负载均衡在文件上传和下载两种场景下的应用。由于这部分的测试用例结果较为复杂，故在此不做用例演示，详情可见 PPT 或演示截图。

3.5.1 负载均衡——下载

基础的负载均衡算法如随机算法、轮询算法，实现非常简单，但没有考虑服务器不同的硬件条件和环境。可以通过加权的方式解决，如三个节点

{a=1,b=2,c=4}，通常最后的调用顺序是 c， c， c， c， b， b， a。

借鉴 Nginx 的做法，使得选择节点平滑，均匀分摊，每个节点有三个权重变量：

1.weight：初始权重，即在配置文件或初始化时每个节点的权重。

2.effWeight：有效权重，初始化为 weight。

- 在通讯过程中发现节点异常，则-1；
- 之后再次选取本节点，调用成功一次则+1，直达恢复到 weight；
- 此变量的作用主要是节点异常，降低其权重。

3.curWeight：节点当前权重，初始化为 0。

该算法首先轮询所有节点，计算当前状态下所有节点的有效权重之和权重；接着计算当前权重=当前权重 + 有效权重，选出所有节点当前权重最大的一个；选中节点后，其当前权重 =当前权重 - 总权重。下表为一个例子，调用顺序为 c b c a c b c，7 次调用后初始权重 curWeight 又回到 c=0,b=0,a=0 的状态。

请求序号	请求前 CurWeight	选中节点	请求后 CurWeight
1	a= 1, b= 2, c=4	c	a= 1, b= 2, c=-3
2	a= 2, b= 4, c=1	b	a= 2, b=-3, c=1
3	a= 3, b=-1, c=5	c	a= 3, b=-1, c= 2
4	a= 4, b= 1, c=2	a	a=-3, b= 1, c= 2
5	a=-2, b=3, c=6	c	a=-2, b= 3, c=-1
6	a=-1, b= 5, c=3	b	a=-1, b=-2, c= 3
7	a= 0, b= 0, c=7	c	a= 0, b= 0, c= 0

表 1 加权轮询算法示例

3.5.2 负载均衡——上传

连接节点信息：节点地址+存储的文件个数，通过节点信息我们可知道每个

节点服务器的负载情况，据此设计负载均衡算法。

算法首先将各节点按照加入服务器的时间进行排序，每个节点都包含 IP 地址和文件数据，记第一个节点的文件数目为 minX。当有新文件要进行上传时，服务器会按照排序依次进行文件数目检测，若小于 minX，则将 minX 更新为该文件数目。若没有节点的文件数目小于 minX，则选择第一个节点进行上传。

例如现要上传一个 upload_test.txt 文件，此处，节点 1 有 4 个文件，节点 2 有三个文件，节点 4 只有 1 个文件。节点 4 的文件数最少，根据负载均衡机制选择节点 4 进行上传。

3.6 容灾备份

容灾备份实际上是两个概念，容灾是为了在遭遇灾害时能保证信息系统能正常运行，帮助企业实现业务连续性的目标，备份是为了应对灾难来临时造成的数据丢失问题。在容灾备份一体化产品出现之前，容灾系统与备份系统是独立的。容灾备份产品的最终目标是帮助企业应对人为误操作、软件错误、病毒入侵等“软”性灾害以及硬件故障、自然灾害等“硬”性灾害。

本系统设计了服务器崩溃处理和文件备份机制。对节点服务器和目录服务器均做了崩溃处理。文件备份也设计了自动备份和手动备份两种机制。由于这部分的测试用例结果较为复杂，故在此不做用例演示，详情可见 PPT 或演示截图。

3.6.1 崩溃处理

服务器崩溃的情况分成两种，一种是客户端与目录服务器在交互的过程中出现崩溃，另一种是客户端与节点服务器交互时出现崩溃，这两种崩溃的处理方法比较相似，都是客户端在向服务器发送请求的时候，采用 try，except，finally 的异常处理机制，捕捉的异常为 BaseException 类型，即包含了所有的异常类型。在 try 中进行正常地服务器访问请求，当出现异常时，即服务器崩溃，客户端与服务器的交互失败，这时启用异常处理中给出的备选方案。

对于目录服务器崩溃，我们使用备用的目录服务器接管请求并处理。客户端向备用的目录服务器进行发送同样的请求，所以在此之前必须确保备用目录服务

器和主目录服务器的目录表以及系统备份文件夹都要时刻保持相同，同时也要求在配置的文件中记录下备用的目录服务器的 IP 地址以及端口号。

以文件下载为例进行简要说明，其他状况同理。首先主服务器和备用服务器都正常运行，随后在客户端将要下载文件时中断主服务运行，此时主服务器崩溃，切换至备用服务器，文件仍然下载成功。

对于节点服务器崩溃，备选方案是在客户端向主目录服务器进行发送文件上传或下载请求时，因为主目录服务器中设定了系统备份的文件夹，所以依然能满足客户端的请求。

以文件上传为例进行简要说明，其他状况同理。首先节点服务器 1 有两个文件，节点服务器 7 只有一个文件，依据前面的负载均衡机制，客户端上传文件应上传至节点 7 中。在客户端将要上传文件时中断节点服务器的运行。此时目录服务器会识别节点 7 服务器发生故障，并将文件上传至系统缓存文件夹。

3.6.2 文件备份

对于文件备份机制，我们的初衷是只提供手动备份功能，这样做的想法源于我们对备份的初步理解是某些文件可能比较重要，那么自然我们要为这些文件提供备份功能。手动备份是指我们为客户端提供了备份选项，是否使用该功能是由用户自我决定的。用户如果觉得某个文件比较重要，会使用对该文件的手动备份选项，目录服务器会根据负载均衡算法选择合适的节点服务器对该文件做备份，并且将该节点信息返回到客户端，客户端会向该节点服务器发送备份请求，该节点服务器会再向目录服务器查询有哪些节点服务器存储该文件，目录服务器会将包含该文件的节点信息返回给欲备份节点服务器。欲备份节点服务器再向该节点服务器申请下载该文件至备份节点服务器本地。

为了提升系统的稳定性和可用性，仅凭手动备份是远远不够的。 对于分布式文件系统来说，对所有文件做一个统一的系统备份是必须的。所以我们设计了自动备份机制。具体做法是为目录服务器提供了系统备份文件夹，它会存储系统所有节点的文件。节点服务器一旦有新文件上传时，节点会主动将该新上传的文件申请备份到系统备份文件夹。同时，一旦有新节点被创建（可能之前崩溃过），

新创建的节点会主动将它本地的所有文件上传至系统备份文件夹。因为系统包含的文件发生改变无外乎这两种情况，所以备份文件夹包含了该系统的所有文件信息。特别地，当发现有同名的文件加入时，会先进行判断，不会加入重复的文件，只有新的文件才会加入到目录服务器的备份文件夹中，这种机制使得文件自动备份更高效。

3.7 读写锁

实现分布式文件系统时，最基本、最重要的原则就是实现分布式文件系统的一致性。实现的标准就是：客户端对文件可以同时读，但不能同时写。由于这部分的测试用例结果较为复杂，故在此不做用例演示，详情可见 PPT 或演示截图。

客户端申请对某文件的读权限时，需要判断该文件有无写锁，若此文件无写锁，即可对文件进行下载读取，并加上读锁，完成对文件的读取后，手动解除该文件上的读锁；客户端申请对某文件的写权限时，需要判断该文件有无写锁或读锁，若此文件无写锁，即可对文件进行下载并修改，同时加上写锁。时间戳之前无读锁，才能提交文件，解除写锁。锁的机制如下表所示。

锁状态	读锁请求	写锁请求
无锁	可以	可以
读锁	可以	无法提交
写锁	阻塞	阻塞

表 2 锁机制状态表

对于读锁，以“读锁可读不可写”为例，其他状况同理。首先客户端 1 以读方式下载文件，同时上读锁。读锁后，客户端 2 可以读该文件。上读锁后，客户端 3 虽可以下载该文件，但是不能再上传该文件，即不能写文件；对于写锁，以“写锁不可读不可写”为例，其他状况同理。首先客户端 1 以写方式下载文件，同时上写锁。上写锁后，客户端 2 无法读该文件。上写锁后，客户端 3 也无法写

该文件。

3.8 版本控制

由于分布式文件系统的高并发性导致文件更迭速度很快，我们便引入版本控制来方便对文件进行管理。但我们并没有做到像 Svn、Git 等追踪文件的变更，比如时间、提交人等信息，我们只是简单地记录每个文件的版本总数量，且只保存当前最新的版本文件。由于这一功能较为简单，所以不做特别演示。

为了实现这一功能，我们为目录服务器设置了一个文件的版本号列表（fileVersion），该列表记录了每个文件的当前版本号。新创建节点的文件版本号初始设为 1，每当上传文件新版本时，都会将该文件的版本号+1。

同时，版本控制的另外一个重要功能是：避免新的低版本的同名文件覆盖节点服务器已存在的文件。在客户端对文件进行上传时，我们对文件的版本进行了比较，上传的文件版本号只有比同名文件版本号更高时，才允许被上传。这样新建文件版本号初始为 1，自然不能覆盖节点服务器的已有同名文件。

四、项目总结

4.1 系统特点

- 访问透明性：采用分层结构，客户端不需要知道上传、下载、备份等操作是向具体的哪个节点服务器进行交互的，实现文件的透明访问。
- 一致性：使用带有时间戳的文件锁机制，确保在整个分布式文件系统中的文件一致性。
- 备份：实现了目录服务器的自动备份以及客户端的手动备份的功能，确保系统的安全可靠，同时改善容错、提高性能。
- 负载均衡：对文件的上传和下载都进行了负载均衡设计，同时还在客户端方面设计了缓存文件夹，确保系统在负载过重的情况下能有效的提高处理效

率。

• 容错性：采用了备用的目录服务器以及系统备份文件夹，确保客户端在与服务器进行交互时，服务器出现崩溃情况后仍然能得到有效处理。

• 高效性：通过下载缓存对重复下载的文件设置了合理机制，避免无必要的重复下载而导致资源浪费。

4.2 未来工作

• 安全性：对不同用户进行登录验证，并设置相应权限，同时对文件传输采用加密形式等。

• 锁机制：读写锁扩展为文件的读写权限管理，由客户端进行手动解锁改进为客户端在读/写操作结束后自动向目录服务器权限管理发送读/写结束信号，从而实现自动解锁。

• 代理管理：引入专门的 agent 来对主目录服务器与备用目录服务器进行管理，主目录服务器与备用目录服务器也进行通信，主目录服务器与备用目录服务器能在某一服务器宕机时智能切换。

• 文件的分块存储：性能上需要对文件的存储采用 Block(块)的模式，即将大量的小文件也就是实际数据文件合并成一个大文件，加快读取吞吐量。

• 负载均衡优化：我们的负载均衡只根据节点的数量进行负载均衡来选择节点，未来将文件的大小也作为衡量指标。

• 文件系统过滤：对申请访问的文件进行权限验证和限流熔断，以提高系统的可用性。

• 并发控制：引入多线程编程技术，可以和文件的分块存储结合提高系统效率，例如对文件的分块实行并发下载，大大提高下载速度的同时减轻节点服务器的负载。

五、课程总结

5.1 课程收获

分布式系统这门课将要结束，本科我们主要学习的独立系统的算法和理论，通过这门课我们也学习了分布式系统的理论和知识，对计算机系统有了更全面、更清晰的认识。在此过程中，我们学习了分布式系统的基本概念、分布式系统时钟、系统模型、处理机调度、传输和通信、选举算法、文件复制等知识。通过老师的讲解，我对这些知识的理解更加深刻，对我日后的学习和工作都帮助很大。在此，感谢何老师一个多月来对我们全体学员的谆谆教诲，这门课程让我受益匪浅、获益一生，再一次感谢何老师您的辛勤付出！

5.2 课程建议

分布式系统是计算机系统中不可或缺的一部分，不论是在科研领域还是工程领域都有着重要意义，围绕分布式系统也有许多的科研工作者和工程师们对其中的问题展开研究，所以我想我们在学习理论知识的同时也可以更多的了解学术界和工业界在分布式系统有关问题上的最新进展，一些实际案例或许对我们的学习更有启发。

参考文献

[1] 张海茹.基于 TFS 的分布式文件存储平台研究与实现[J].软件工程与应用,2016,5(03):198-203. DOI:10.12677/SEA.2016.53022

[2] 孙耀，刘杰，叶丹，钟华.分布式文件系统元数据服务的负载均衡框架[J].软件学报,2016,27(12):3192-3207.

[3] 卢淑君.云存储中分布式文件系统 TFS 的改进研究[D].江苏:南京邮电大学,2017.

[4] 负载均衡算法[EB/OL].https://www.cnblogs.com/markcd/p/8456870.html

[5] 分布式文件系统[EB/OL].https://github.com/donaltuohy/DFS

[6] GeorgeCoubouris, JeanDollimore, TimKindberg, et al. 分布式系统概念与设计[M]. 机械工业出版社, 2003 P308-309.

[7] 百度百科 [EB/OL].https://baike.baidu.com