# 基于 FastDFS 文件系统的理解

## 目录

# 摘要

随着信息技术的飞速发展，大量的数据需要进行存储，为了解决这种问题，企业往往需要购买大量的存储设备，这不但需要很高的成本，而且大量的数据管理很麻烦，不容易维护，用这种传统的方式就会暴露出种种弊端。云存储作为一种随着云计算而发展出来的技术，其关注的中心恰恰是海量数据的存储和管理，同时因为具备弹性扩展、方便海量数据管理以及低成本等优点。

FastDFS 是一款有国人研发并且开放源码的分布式文件系统，在类 UNIX 系统上能很好的工作。对比其他分布式文件系统，它的优势是量级轻，满足高并发访问的需求，容易扩展，具备负载均衡功能，并且能自动进行文件同步。另外对小文件的存储性能较好。

本文首先对研究背景做出了叙述，分析了引入 FastDFS 意义。接下来对 FastDFS 的理论知识进行探究，主要对云储存的简介、FastDFS 的使用前提、FastDFS 的架构分析、FastDFS 各模块的功能逻辑分析以及几种分布式文件系统做出了对比。后面两章是对 FastDFS 的集群同步和小文件储存两大重要功能做出了研究和分析。

通过本文对 FastDFS 的分析，我们可以了解目前分布式文件系统设计的基本原理、集群之间数据是如何同步的以及掌握 FastDFS 对小文件的存储方案。



关键词：存储；分布式系统；FastDFS

# 第一章 研究的背景与意义

随着信息技术的迅猛发展与普及，越来越多的人开始使用网络。 随之不断增长的还有数据存储的量级。在这种情况下，依靠传统的存储方式，不但成本过高，而且数据管理复杂，效率低。因此，研究如何高效的存储和管理大量的数据，降低硬件成本，提供更好的数据访问服务，对于企业以及个人有积极的意义。

首先，降低成本。云存储使用大量廉价的设备存储数据，对于企业，减少了购买昂贵存储服务器的成本，因为传统的集中式存储，为了确保数据的容错性，需购买专用设备，企业需要投入大量资金，因此也限制了用户的存储空间，如早期的邮箱服务，用户的存储空间上限仅 2G，云存储技术的应用，使用户的存储空间大大提高，享受到更好的服务。从另一方面来说，由于采用了廉价的存储设备，企业对设备的维护以及管理成本也大幅度降低。

其次，易于管理。云存储在设计时就考虑了数据的管理，特别是海量数据的管理，通过使用虚拟化技术，将设备虚拟化成资源池，通过资源池实现对数据的管理，可以方便的完成数据的备份以及迁移等操作。传统的存储虽然也可以管理海量数据，但是对数据的管理和维护相当复杂。

再者，灵活可靠。云存储可以根据用户的需求，灵活分配用户的存储空间，实现量身定制。当存储空间不足时，可以采用热插拔的方式增加存储设备，扩展方便。云存储自动对数据实行备份，在发生数据灾难的情况下，可以迅速恢复。

分布式文件系统是云存储的重要技术，良好的文件系统不但需要支持海量数据的存储，方便扩展，而且具有高可用，在某些节点失效的情况下仍能提供服务，并且易于管理使用。FastDFS 作为国人自助研发的分布式开源文件系统，具备良好的性能，满足云存储的要求，可以将其应用到云存储之中。对于用户来说，不必关心数据存储的具体细节，只需要将网络接入系统，就能随时地访问到数据了。

# 第二章 理论知识的探究

## 2.1 云存储的简介

云存储是伴随云计算而兴起的一个技术。通过运用集群、网格技术、虚拟化以及分布式文件系统等方式，将不同类型的存储设备结合起来对外提供数据存储和访问。对于用户而言，不必考虑底层的存储设备以及实现细节。系统会将用户的数据进行存储、归档、备份，实现对数据的使用、共享以及保护的目的。云存储不但是一种技术，而且是一种服务。用户只需要通过网络连接到云端，便可以随时随地的访问自己的数据。

相对于传统的存储方式，云存储有显著的优势。首先，云存储具备较高的可扩展性，支持线性扩容，相对于传统方式在管理海量数据上更方便。其次具有较好的可用性，用户只需要拥有网络就可以随时随地的访问数据，不受设备、地点的限制。再者，云存储能减少存储的成本，相对于传统存储方式，存储成本和维护成本大幅度的降低。

## 2.2 FastDFS 使用前提

使用 FastDFS 的前提：访问量大，数据量大。

传统的企业级开发对于高并发要求不是很高，而且数据量可能也不大，在这样的环境下文件管理可能非常 Easy。

但是互联网应用访问量大、数据量大，在互联网应用中，我们必须考虑解决文件大容量存储和高性能访问的问题，而 FastDFS 就特别适合干这件事情，常见的图片存储、视频存储、文档存储等等我们都可以采用 FastDFS 来做。

## 2.3 FastDFS 架构分析

FastDFS 是由淘宝资深架构师余庆开发，是一个功能比较完善的分布式存储框架。

FastDFS 主要的功能包括：文件存储，同步和访问，设计基于高可用和负载均衡。FastDFS 非常适用于基于文件服务的站点。

Fast 是由跟踪服务器（tracker server），存储服务器（storage server）和客户端（client）。特别适合于中小文件（文件大小：4KB-500MB）海量数据存储问题，例如图片分享和视频分享网站。

FastDFS 的架构图如图 2-1 所示：

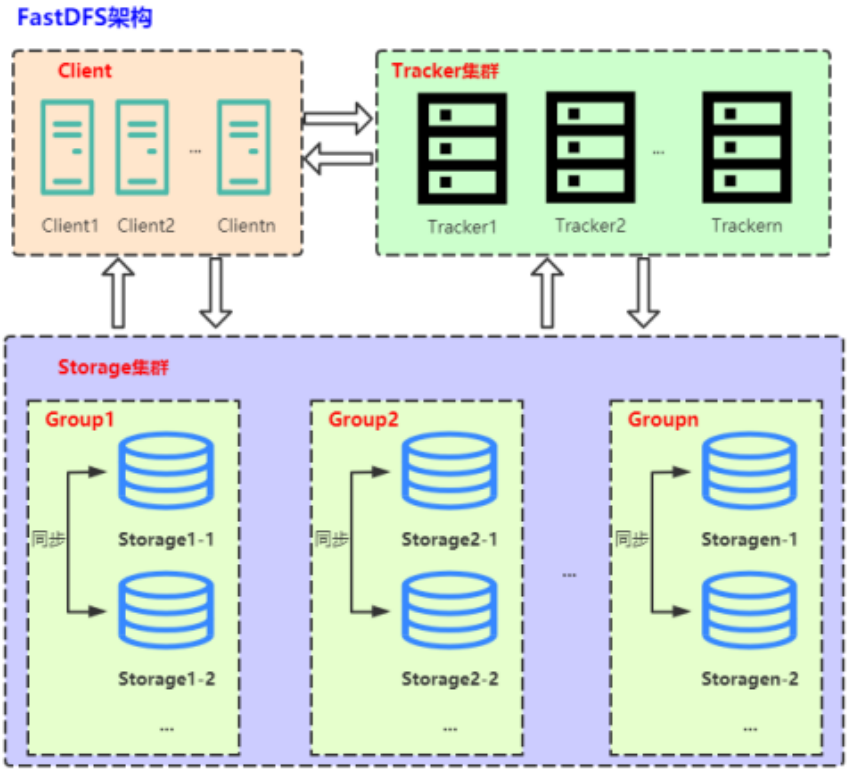


图 2-1 FastDFS 系统架构图

### 2.3.1 Tracker server

Tracker 是 fastdfs 的协调者，负责管理所有的 storage server 和 group，每个 storage 在启动后会连接 tracker，告知自己所属的 group 等信息，并保持周期性心跳，tracker 根据 storage 的心跳信息，建立 group==>【storage server list】的映射表。

Tracker 需要管理的元信息很少，会全部存储在内存中，里面的信息都是由 storage 汇报的信息生成的，本身是不需要持久化任何数据，这就使得 tracker 非常容易扩展，直接增加 tracker 机器即可扩展为 tracker cluster（集群）来服务,cluster 里每个 tracker 之间是完全对等,所有的 tracker 都接受 storage 的心跳信息，生成元数据来提供读写服务。

### 2.3.2 Storage server

Storage server（后面简称为 storage）以组（卷，group 或 volume）为单位组织，一个 group 内包含多台 storage 机器，数据互为备份，存储空间以 group 内容量最小的 storage 为准，故配置多个 storage 尽量硬件资源相同，以免造成浪费。

以 group 为单位租住的存储能方便进行隔离，负载均衡，副本数定制（group 内 storage 的数量为该 group 的副本数），比如将不同引用数据存到不同的 group 就能隔离应用数据，同时还能根据应用的访问特性来将应用分配到不同的 group 来做负载均衡；缺点就是 group 的容量受单机存储容量的限制，同时当 group 内有机器坏掉时，数据恢复只能依赖 group 内的其他机器，使得恢复时间会很长。

group 内每个 storage 的存储依赖于本地文件系统，storage 可配置多个数据目录，比如有 10 快磁盘，分别挂载在/data/disk1-/data/disk10, 则可将这 10 个目录配置为 storage 的数据存储目录；

storage 接受到写文件请求时，会根据配置好的规则，选择其中一个存储目录来存储文件。为了避免单个目录下的文件太多，在 storage 第一次启动时，会在每个数据存储目录中创建 2 级子目录，每级 256 个，总共 65536 个文件，新写的文件会以 hash 的方式被路由到其中某个子目录下，然后将文件数据直接作为一个本地文件存储到该目录中。

### 2.3.3 Client

FastDFS 向使用者提供基本文件接口，比如 monitor、upload、download、append、delete 等，以客户端库的方式提供给用户使用。

## 2.4 FastDFS 各功能逻辑分析

### 2.4.1 upload file 原理图如图 2-2:

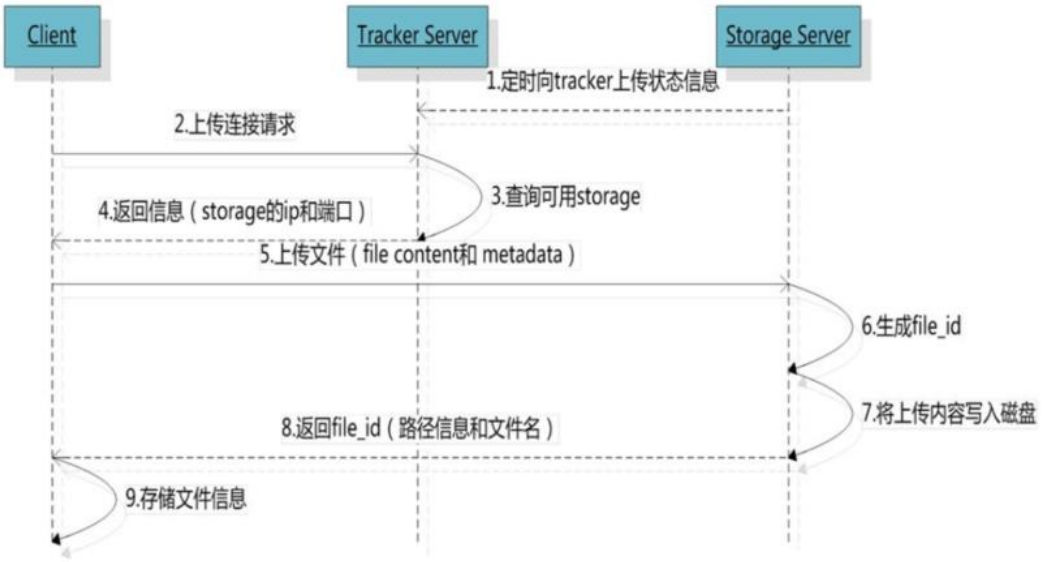


图 2-2 FastDFS 上传文件流程

1. 选择 tracker server

   当集群中不止一个 tracker server 时，由于 tracker 之间是完全对等的关系，客户端在 upload 文件时可以任意选择一个 tracker。

2. 选择存储的 group

   当 tracker 接收到 upload file 请求时，会为该文件分配一个可以存储该文件的 group，支持如下选择 group 的规则：

   a) Round robin ， 所有的 group 轮询
   b) Specified group，指定某一个确定的 group
   c) Load balance 选择最大剩余空间的组上传文件

3. 选择 storage server

   当选定 group 后，tracker 会在 group 内选择一个 storage server 给客户端，支持如下选择 storage 的规则：

   a) Round robin ，在 group 内的所有 storage 轮询
   b) First server order by ip，按照 ip 排序
   c) First server order by priority，按优先级排序（优先级在 storage 中配置）

4. 选择 storage path

   当那个分配好 storage server 后，客户端将向 storage 发送写文件请求，storage 将会为文件分配一个数据存储目录，支持如下规则：

   a) Round robin，多个存储目录间轮询
   b) 剩余存储空间最多的优先

5. 生成 Fileid

选择存储目录之后，storage 会为文件生一个 Fileid，由：storage server ip、文件创建时间、文件大小、文件 crc32 和一个随机数拼接而成，然后将这个二进制串进行 base64 编码，转换为可打印的字符串。

6. 选择两级目录

当选定存储目录之后，storage 会为文件分配一个 fileid，每个存储目录下有两级 256*256 的子目录，storage 会按文件 fileid 进行两次 hash，路由到其中一个子目录，然后将文件以 fileid 为文件名存储到该子目录下。

7. 生成文件名

当文件存储到某个子目录后，即认为该文件存储成功，接下来会为该文件生成一个文件名，文件名由：group、存储目录、两级子目录、fileid、文件后缀名（由客户端指定，为了区分文件类型）拼接而成的，如图 2-3。


图 2-3 生成文件名构成

文件名规则：

a) Storage_id（ip 的数值类型）源 storage server ID 或 IP 地址
b) Timestamp（文件创建时间戳）
c) File_size(若原始值为 32 位则前面加入一个随机值填充,最终为 64 位）
d) Crc32 （文件内容的校验码）

随机数（引入随机数的目的是为了防止生成重名文件）

## 2.4.2 download file 原理图如图 2-3

客户端 upload file 成功后，会拿到一个 storage 生成的文件名，接下来客户端根据这个文件名即可访问到该文件，跟 upload file 一样，在 download file 时客户端可以选择任意 tracker server。

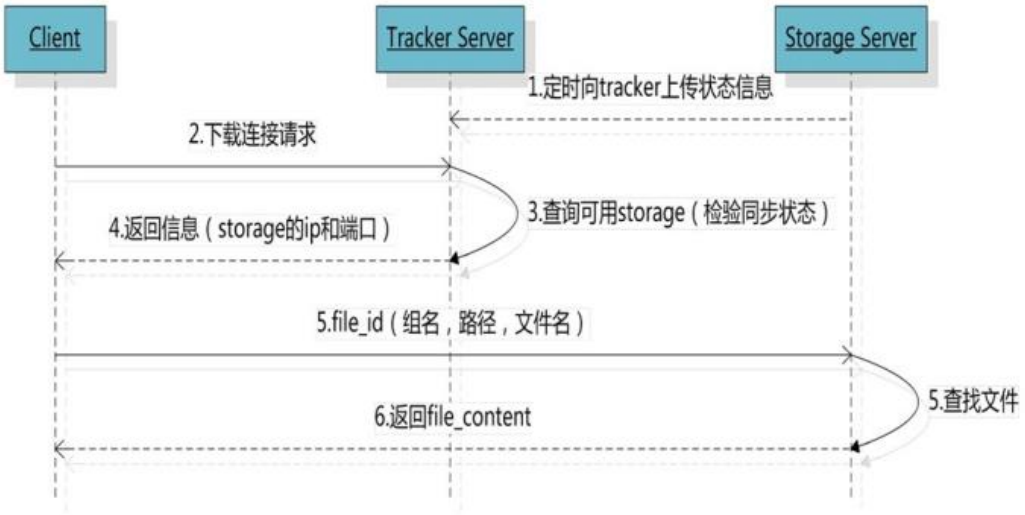


图 2-3 FastDFS 下载文件流程

Tracker 发送 download 请求给某个 tracker，必须带上文件名信息，tracker 从文件名中解析出文件的 group、大小、创建时间等信息，然后为该请求选择一个 storage 用来服务读请求。由于 group 内的文件同步时在后台异步进行的，所以有可能出现读到时候，文件还没有同步到某些 storage server 上，为了尽量避免访问到这样的 storage，tracker 按照如下规则选择 group 内可读的 storage。

1. 该文件上传到的源头 storage - 源头 storage 只要存活着，肯定包含这个文件，源头的地址被编码在文件名中。
2. 文件创建时间戳==storage 被同步到的时间戳且（当前时间 - 文件创建时间戳）> 文件同步最大时间（如 5 分钟）- 文件创建后，认为经过最大同步时间后，肯定已经同步到其他 storage 了。
3. 文件创建时间戳 < storage 被同步到的时间戳。同步时间戳之前的文件确定已经同步了。
4. （当前时间 - 文件创建时间戳）> 同步延迟阈值（如一天）。经过同步延迟阈值时间，认为文件肯定已经同步了。

### 2.4.3 HTTP 下载逻辑

FastDFS 自带的 http 服务已经弃用了，目前是需要通过 nginx + fastdfs-nginx-module 的方式去实现下载。

## 2.5 几种 DFS 对比如下

Ceph, TFS, FastDFS, MogileFS, MooseFS, GlusterFS 对比如下图 2-4，

| 对比说明<br>/文件系统 | TFS | FastDFS | MogileFS | MooseFS | GlusterFS | Ceph |
| --- | --- | --- | --- | --- | --- | --- |
| 开发语言 | C++ | C | Perl | C | C | C++ |
| 开源协议 | GPL V2 | GPL V3 | GPL | GPL V3 | GPL V3 | LGPL |
| 数据存储方式 | 块 | 文件/Trunk | 文件 | 块 | 文件/块 | 对象/文件/块 |
| 集群节点通信协议 | 私有协议（TCP） | 私有协议（TCP） | HTTP | 私有协议（TCP） | 私有协议（TCP）/ RDAM (远程直接访问内存) | 私有协议（TCP） |
| 专用元数据存储点 | 占用NS | 无 | 占用DB | 占用MFS | 无 | 占用MDS |
| 在线扩容 | 支持 | 支持 | 支持 | 支持 | 支持 | 支持 |
| 冗余备份 | 支持 | 支持 | - | 支持 | 支持 | 支持 |
| 单点故障 | 存在 | 不存在 | 存在 | 存在 | 不存在 | 存在 |
| 跨集群同步 | 支持 | 部分支持 | - | - | 支持 | 不适用 |
| 易用性 | 安装复杂，官方文档少 | 安装简单，社区相对活跃 | - | 安装简单，官方文档多 | 安装简单，官方文档专业化 | 安装简单，官方文档专业化 |
| 适用场景 | 跨集群的小文件 | 单集群的中小文件 | - | 单集群的大中文件 | 跨集群云存储 | 单集群的大中小文件 |

图 2-4 各种文件系统对比

下方是参考另外的一个对比：

**GFS**: Google FileSystem，分布式系统的开山鼻祖，由于 Google 内部需要遂开发，后来发布论文公布其技术细节，但是没有开源，擅长处理单个大文件。

**HDFS**: Hadoop Distribution FileSystem，GFS 的山寨版，擅长处理单个大文件。

GFS 和 HDFS 都将元数据存储于内存中， 定期存储在持久存储中，只适合存储百万、千万级别的大文件。

**TFS**: TaoBao FileSystem，基于 HDFS 开发 适用于存储海量小文件。

**GlusterFS**: 去中心化设计，没有元数据节点。

**Ceph**: Linux 内核级实现的文件系统， 已经收录 Linux 内核，是一个 Linux PB 级别的分布式文件系统。

**MogileFS**: 适用于存储海量小文件， 使用 perl 语言编写，国内有人使用 C 语言重写并开源为 FastDFS。

**MFS**：MooseFS，通用简便，适用于研发能力较弱的公司。

**FastDFS** 是基于互联网应用的开源分布式文件系统，主要解决了大容量的小文件存储和高并发访问的问题，具有轻量级、支持高并发访问、高可扩展性等优点。

# 第三章 集群同步规则原理

## 3.1 环境部署

按照部署 2 个 tracker server，2 个 storage server 模拟实际情况存储，如表 3-1，

注意事项：多个 tracker 可以部署在同一台机器上，但 storage 不能部署在同一台机器上。

表 3-1: 集群的配置信息

| 服务器地址 | 服务程序 | 对应配置文件（端口区分） |
|---|---|---|
| 192.168.254.131 | fdfs_trackerd | tracker_22122.conf |
| 192.168.254.131 | fdfs_trackerd | tracker_22123.conf |
| 192.168.254.131 | fdfs_storaged | storage_group1_23000.conf |
| 192.168.254.132 | fdfs_ storaged | storage_group1_23000.conf |

1. 需要在同一个服务器上创建多个 tracker 存储路径
2. 需要把现有的 storage 和 tracker 全部停止 fdfs_trackerd /etc/fdfs/tracker_22124.conf stop
3. 需要修改对应的配置文件 tracker_22122.conf 和 tracker_22123.conf 的存储日志和数据的路径。
4. 修改 254.131 机器和 254.132 机器的 storage 配置文件
5. 修改 client.conf 的配置文件，添加 tracker_server 的地址等
6. 检查是否正常启动，分别在两台服务器上执行：
/usr/bin/fdfs_monitor /etc/fdfs/storage_group1_23000.conf
如果成功，正常两边都提示如图 3-1 内容，表示两边都存在 2 个 active 的 storage。

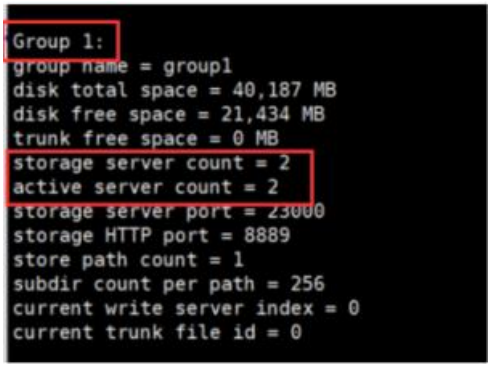


图 3-1: 提示成功的信息

## 3.2 binlog 格式

FastDFS 文件同步采用 binlog 异步复制方式。storage server 使用 binlog 文件记录文件上传、删除等操作，根据 binlog 进行文件同步。Binlog 中只记录

文件 ID 和操作，不记录文件内容。下面是几行 binlog 文件内容示例如图 3-2：

```
1572660675 C M00/00/00/oYYBAF285cOlHiVCAACI-7zX1qUAAAAVgAACC8AAlkT490.txt
1572660827 c M00/00/00/oYYBAF285lulK8jCAAAJeheau6AAAAAVgABI-cAAAmS021.xml
1572660911 D M00/00/00/oYYBAF285cOlHiVCAACI-7zX1qUAAAAVgAACC8AAlkT490.txt
1572660967 d M00/00/00/oYYBAF285lulK8jCAAAJeheau6AAAAAVgABI-cAAAmS021.xml
```

图 3-2 binlog 格式

从上面可以看到，binlog 文件有三列，依次为：

1. 时间戳
2. 操作类型
3. 文件 ID（不带 group 名称）

   Storage_id(ip 的数值型)

   Timestamp（创建时间）

   File_Size（若原始值为 32 位则前面加入一个随机值填充，最终为 64 位）

   Crc32（文件内容的校验码）

文件操作类型采用单个字母编码，其中源头操作用大写字母，被同步的操作为对应的小写字母。文件操作字母的含义为图 3-3 所示：

| 源 | 副本 |
| --- | --- |
| C：上传文件（upload） | c：副本创建 |
| D：删除文件（delete） | d：副本删除 |
| A：追加文件（append） | a：副本追加 |
| M：部分文件更新（modify） | m：副本部分文件更新（modify） |
| U：整个文件更新（set metadata） | u：副本整个文件更新（set metadata） |
| T：截断文件（truncate） | t：副本截断文件（truncate） |
| L：创建符号链接（文件去重功能，相同内容只保存一份） | l：副本创建符号链接（文件去重功能，相同内容只保存一份） |

图 3-3 文件操作字母对应含义

注：源表示客户端直接操作的那个 storage 即为源，其他的 storage 都为副本

## 3.3 同步规则

1. 只在本组内的 storage server 之间进行同步；
2. 源头数据才需要同步，备份数据不需要再次同步，否则就构成环路了，源数据和备份数数据区分是 binlog 的操作类型来区分，操作类型是大写字母就表示源数据，小写表示备份数据；
3. 当新增加一台 storage server 时，由已有的一台 storage server 将已有的所有数据（包含源数据和备份数据）同步给该新增服务器。

## 3.4 binlog 同步过程的流程

在 FastDFS 之中，每个 storage 之间的同步都是由一个独立线程负责的，该线程中的所有操作都是以同步方式执行的。比如一组服务器有 A,B,C 三台机器，那么在每台机器上都有两个线程负责同步，如 A 机器，线程 1 负责同步数据到 B，线程 2 负责同步数据到到 C。

# 第四章 小文件存储

## 4.1 小文件存储的背景

通常我们认为大小在 1MB 以内的文件称为小文件，百万级数量及以上称为海量，由此量化定义海量小文件问题，后面简称 LOSF。LOSF 应用在目前实际中越来越常见，如社交网站、电子商务、网络视频、高性能计算，这里举几个典型应用场景。著名的社交网站 Facebook 存储了 600 亿张以上的图片，推出了专门针对海量图片定制优化的 Haystack 进行存储。淘宝目前应该是最大 C2C 电子商务网站，存储超过 200 亿张图片，平均大小仅为 15KB，也是推出了针对小文件优化的 TFS 文件系统存储这些图片，总图片数量能够超过 20 亿；视频会由切片服务器根据视频码切割成 1MB 左右的分片文件，100 个频道一个星期的点播量，分片文件数量可达到 1000 万量级。动漫渲染和影视后期制作应用，会使用大量的视频、音频、图像、纹理等原始素材，一步普通的动画电影可能包含超过 500 万的小文件，平均大小在 10-20KB 之间；金融票据影像，需要对大量原始票据进行扫描形成图片和描述信息文件，单个文件大小为 5-100KB 的不等，文件数量达到达到数千万乃至数亿，并且逐年增长。

## 4.2 小文件存储性能优化

小文件的性能瓶颈主要来自对于元数据服务器（如 fastdfs 中的 tracker server）的访问，因为当文件本身大小很小时，元数据存储所占空间与文件内容存储所占空间的比利就变得较大，访问元数据所消耗资源与访问文件内容所消耗的比利也变得较大。因此，通常对于小文件存储的优化方法主要有两大类思路：一是减少访问元数据的次数，比如 cache 预取；二是减少元数据所占的存储空间，比如 FastDFS 使用的文件名策略。

## 4.3 FastDFS 文件名策略

FastDFS 中的文件名是在向 StorageServer 存储文件时由系统指定的，文件名中包含了 VolumeID 和 FileID。也就是说，当客户要读取某个文件时，通过

在客户端对文件名进行解析，就可以知道该文件存储在哪个 Volume 上和它在 StorageServer 中的 FileID 。但是此时用户还不能读取文件，因为他不知道 Volume 内各个 StorageServer 的 ip 地址，也不知道应该从 Volume 内的哪个 StorageServer 中读取。所以用户需手持欲访问的文件的 VolumeID 向 TrackerServer 询问，TrackerServe 会均衡当前各 StorageServer 的 IO 负载状况,返回一个最佳的 StorageServer 的 ip 地址。最后用户与该 StorageServer 连接，出示欲访问文件的 FileID ，StorageServer 上会维持一个 FileID 对应偏移量的表，从而得到欲访问文件的偏移量。

可见，FastDFS 的文件名策略将文件存储位置信息隐含在文件名中，从而减少了元数据量，达到了优化小文件存储性能的作用。

## 4.4 小文件的配置文件

在/etc/fdfs/tracker.conf 中查找相应字段：
fastDFS 中的默认是关闭 trunk 存储的
use_trunk_file = false;
slot_min_size = 256
slot_max_size = 1MB

## 4.5 合并存储文件命名与文件

向 FastDFS 上传文件成功时，服务器返回该文件的存取 ID 叫做 fileid，当没有启动合并存储时该 fileid 和磁盘上实际存储的文件一一对应，当采用合并存储时就不再一一对应而是多个 fileid 对应的文件被存储成一个大文件。
注：下面将采用合并存储后的大文件统称位 Trunk 文件，没有合并存储的文件统称位源文件。
下面有三个概念：
1. Trunk 文件：storage 服务器磁盘上存储的实际文件，默认大小位 64MB
2. 合并存储文件的 Fileid：表示服务器启用合并存储后，每次上传返回给客户端的 Fileid，注意此时该 Fileid 与磁盘上的文件没有一一对应关系；
3. 没有合并存储的 Fileid：表示服务器未启用合并存储时，Upload 时返回的 Fileid。
Trunk 文件名格式： storage1/data/00/00/000001 文件名从 1 开始递增，类型为 int。
例如：
未合并的：group1/M00/00/00/wKj-g2C_cx2ASZeyAAAAG_r9TCQ854.txt
合并的：
group1/M00/00/00/wKj-g2C_cLOIMi0sAAAAA_2i0i4AAAAAQAAAAAAAAEA766.txt
两者的区别就是在启动合并存储时服务返回给客户端的 fileid 有变化：

### 4.5.1 没有合并存储时

1. 没有合并存储时 fileid ： 文件名（不含后缀名）采用 Base64 编码，包含下面 5 个字段（每个字段均为 4 字节整数）

group1/M00/00/00/wKj-g2C_cx2ASZeyAAAAG_r9TCQ854.txt

2. 这个文件名中，除了.txt 为文件后缀，前面的部分是一个 base64 编码缓冲区，组成如下：

Storage_id（ip 的数值类型）源 storage server ID 或 IP 地址

Timestamp（文件创建时间戳）

File_size（若原始值为 32 位则前面加入一个随机值填充，最终为 64 位）

Crc32 （文件内容的校验码）

随机数（引入随机数的目的是为了防止生成重名文件）

### 4.5.2 合并存储时 fileid

1. 如果采用合并存储，生成的文件 ID 将变长，文件名后面多了 base64 文本长度 16 字符（12 个字节）。这部分同样采用 Base64 编码，包含如下 3 个字段（每个字段均为 4 字节整数）

group1/M00/00/00/wKj-
g2C_cLOIMi0sAAAAA_2i0i4AAAAAQAAAAAAAAEA766.txt

2. 采用合并的文件 ID 更长，因为其中需要加入保存的大文件 id 以及偏移量，具体包括了如下信息：

File_size: 占用大文件的空间

Mtime：文件修改时间

Crc32: 文件内容的校验码

Formatted_ext_name: 文件扩展名

Alloc_size: 分配空间，大于或等于文件大小

Trunk file ID: 大文件 ID 如 000001

Offset：文件内容在 trunk 文件中的偏移量

Size: 文件大小，真正的文件大小，file_size 则为占用的空间大小

### 4.5.3 Trunk 文件内部结构

Trunk 内部是由多个小文件组成，每个小文件都会有一个 trunkHeader，以及紧跟在其后的真实数据，结构如图 4-1 所示：

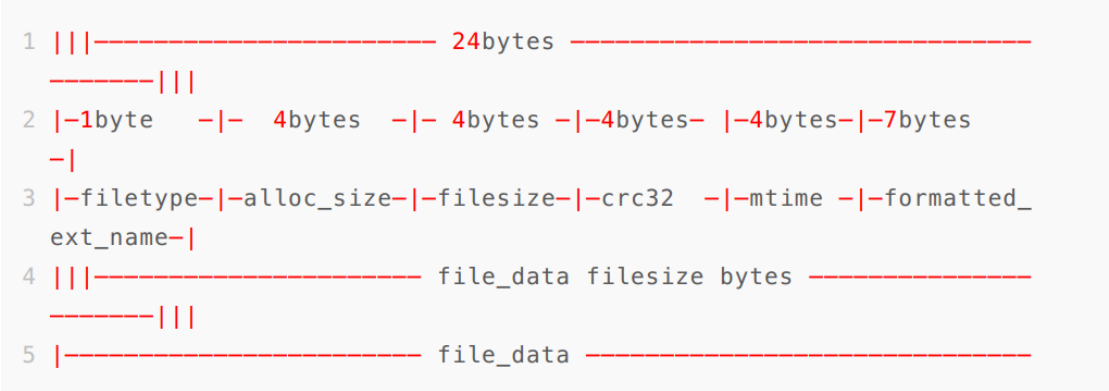


图 4-1：trunk 文件内部结构

Trunk-Hreader 固定占用了 24 字节。

Trunk 文件为 64MB（默认），因此每次创建一次 Trunk 文件总是会产生空余空间，比如为存储一个 10MB 文件，创建一个 Trunk 文件，那么就会剩下接近 54MB 的空间（TrunkHeader 会用 24 字节，后面为了方便叙述暂时忽略其所占空间），下次要想再次存储 10MB 文件时，就不需要创建新的文件，存储在已经创建的 Trunk 文件中即可。另外当删除一个存储文件时，也会产生空余空间。

在 storage 内部会为每个 store_path 构造一颗以空闲块大小作为关键字的空闲平衡树，相同大小的空闲块保存在链表之中。每当需要储存一个文件时会首先到空闲平衡树中查找大于并且最接近的空闲块，然后试着从该空闲块中分割出多余的部分作为一个新的空闲块，加入到空闲平衡树中。例如：要求储存的文件为 300KB，通过空闲平衡树找到一个 350KB 的空闲块，那么就会将 350KB 的空闲块分裂成两块，前面的 300KB 返回用于储存，后面的 50KB 则继续放置到空闲的平衡树之中。假若此时找不到可满足的空闲块，那么就会创建一个新的 trunk 文件 64MB，将其加入到空闲平衡树之中，再次执行上面的查找操作（此时总是能满足了）。

# 第五章：总结

通过上述对 FastDFS 的研究和分析，基本可以掌握目前分布式存储的基本原理、分布式文件之间的各种功能逻辑关系、知道 FastDFS 在集群中数据同步的原理及流程关系、也能学到 FastDFS 是如何对小文件储做优化的。