# 分布式文件系统FastDFS的实现与研究

## 绪论和导言

### 什么是分布式文件系统

分布式文件系统是一个基于客户端/服务器的应用。这个应用可以使得客户端可以访问服务器就好像是访问本地磁盘一样方便快捷。当一个用户访问服务器上的文件时，服务器会发送给客户端这个文件的一份拷贝，这个文件拷贝会缓存在用户的电脑中，使得用户能够修改和访问文件中的数据，当对文件拷贝的操作结束后，这份文件拷贝被送回服务器取代原来的版本。

因为可能会有多个用户同时访问同一个文件，服务器必须有一个同步机制来确保每个用户访问的都是数据的最新版本，同时，服务器也可能需要锁机制来防止多个用户写和读文件。

Google文件系统(GFS)，微软分布式文件系统，MooseFS，FastFDS，TFS，HDFS(Hadoop)等等都是分布式文件系统的例子。GFS被用在Google的搜索系统中，TFS被用在淘宝网的网站架构中，而FastFDS被使用在诸多的网盘，浏览器软件，和国产网站中。

### 为什么要研究分布式文件系统

随着互联网技术的不断发展，用户的需求逐渐从以前的静态网络变到了如今的动态网络，从以前的同步请求到了现在的异步请求，从以前的单进程到了现在的多线程和线程池，网页也从单纯的阅读资料变成了多媒体、社交、购物、存储等的渠道。互联网可以实现越来越多的功能来满足日益增长的需求，但是这些多功能的实现需要巨大的存储容量和高性能的计算能力，如果单纯的按照以前以单个服务器为主存储一个网站的所有内容，成本非常高，而且不一定能满足网站所需要的性能。分布式文件系统应运而生，最早大放异彩的地方就是google的GFS，Google公司在初创期使用廉价的机器组成集群来满足搜索的需要，降低成本的同时也把工作重心更多地放在了业务需求上，实现了许多高性能的分布式算法和分布式框架，也带给了我们如今高效的Google搜索。分布式文件系统同时也适用于大数据的处理，挖掘大数据背后的联系与用户需求，十分适合用于研究用户行为和创造产品价值，符合未来技术发展的潮流。分布式文件系统也可以用作提供云服务，降低每台PC配置的同时，使每个用户都能享受到高效，迅速，高性能的计算服务。同时，分布式文件系统很适用于初创公司，使得他们可以降低硬件成本并且把重心更多地放在业务逻辑上，为用户带来更方便的用户体验。

### 为什么要研究FastFDS

研究了一些分布式文件系统的资料，首先想到的就是大名鼎鼎的Google File System (GFS),研究了GFS的相关论文发现，GFS的侧重点在于大文件的处理。比方说，GFS系统中的文件快规模大概是64MB，而对比早期的单机文件系统像linux上的FAT32系统（现在也在使用）的块规模却小到4kb。为什么GFS需要这么大的块文件规模呢? 是因为GFS主要针对的需求是大文件的存储与交互，因此，大的块规模可以减少Client和Master的交互，减少master的负担。同时，大的块规模使得client在一个块上进行需要的操作的可能性有所增加，所以client会和chunkserver保持一个较长时间的tcp连接，减少网络负担。但是针对实验环境而言，作者硬件条件有限，无法模拟大容量系统的环境。其次，GFS采用的是单点Master的结构，瓶颈性能在于Master，容易发生单点故障，这是一个在其他的分布式文件系统中已经可以得到解决的问题，因此研究其他文件系统像FDFS更能从多角度多方位探讨分布式文件系统的机理，同时也可以满足当下工程环境对于分布式文件系统的需要。除此之外，FastDFS是一个开源的分布式文件系统，作者可以深入源码来研究其中的内部机理，从而来探讨FastDFS对于分布式文件系统机能的满足情况。同时，FDFS非常适合处理移动互联网应用的情况，因为它的存储对象主要是小型文件（文件大小在4KB到500MB之间），对于当下的互联网移动端的发展也具有更加实用的意义。但是FDFS也有它自身不足的情况，因为小型文件对象和它的轻量级特性决定了FDFS并不适用于分布式计算的场合，然而对于满足分布式计算的系统可能又是一个更广大和更深入的话题了，再讨论这种情况与本文的研究目的不相符合。因此，对FastDFS的研究非常适合于作者的现实情况，也适合全方位地研究一个分布式系统的实现

## FastDFS原理和安装

### 实现原理

服务器角色：跟踪器(tracker)+存储结点(storage)

由于分布式文件系统需要对用户提供文件一致性的服务，因此服务器端需要实现两个基本功能：一个是存储，一个就是对存储服务器的管理。实现这两个大功能的同时也需要考虑到很多具体的细节，比如管理服务器和存储服务器的连接，存储服务器的命名映射，存储服务器的错误恢复等等。但是这个部分所述的原理实现只会涉及到FastDFS的总体的框架，对于它的其他亮点的细节会在下文具体写出。

在分布式文件系统中，实现存储功能的服务器会有很多种叫法，比如chunkserver(GFS, MooseFS文件系统)，Datanode(Hadoop文件系统)，DataServer(TFS文件系统)和FastDFS中的Storage服务器。它们本质上都是为了实现存储这个功能而服务的。相对应的，对于管理这些存储服务器的服务器也会有很多名称但是功能需求一致的例子，比如master服务器(GFS, MooseFS文件系统)，Namenode(Hadoop文件系统)，Nameserver(TFS文件系统)和FastDFS中的Tracker服务器。存储服务器需要完成存储文件和文件管理的功能。这些功能又可以被细化为存储，同步和提供存取接口(upload、download、append、delete等)三个部分。而管理服务器主要完成的功能是调度，管理和负载均衡。

值得注意的是，分布式文件系统中还有一个概念在大多数系统中都是用同一个名字进行表示的，那就是元数据metadata。Metadata 的含义就是描述数据的数据，在被应用到分布式文件系统之前，早就在数据库管理和数字图书馆管理中得到了广泛的应用，因此它是一个跨领域的概念，所以我们可以在众多不同的分布式文件系统中发现这样一个共同的特点。FastDFS的Storage服务器会负责维护文件的metadata，这也类似于Unix系统中的i结点。但是FastDFS是一个轻量级的文件管理系统，它对元数据的管理还是基于服务器OS的文件管理系统之上的。在FastDFS中，Metadata主要以键值对的方式进行储存，比如{width:685,height:285}这样的键值对，一个文件的metadata中可以包含多个键值对，FastDFS对文件的描述是可以自定义扩展的。对比之下，FastDFS的元数据的实现是相当简单的，它并没有标准的格式定义(像GFS那样)，元数据的结构均可由用户自定义来实现。这样的设计使得它减少了管理服务器的负担，因为元数据是保存在存储服务器上，利用本地操作系统的文件系统来调用管理文件。而GFS的元数据包含了三种类型： 文件和块的命名空间，文件到块的映射和块的副本的位置。而这三类元数据都要由Master服务器整体维护的，存储在Master服务器内存中，并用操作日志来进行metadata修改的登记，操作日志也会有备份。相比之下，FastDFS就极大的减少了占用内存的空间，也降低甚至消除了GFS的Master服务器单点故障的缺点，因为FastDFS的tracker服务器也具有分组性和对等性。

FastDFS的架构如下图所示：

计算机生成了可选文字:
Client 
Client 
Client 2 
Tracker cluster 
Tracker 2 
Client M I 
Volume 2 
Storage server 21 
Storage server 22 
Storage server 2 Y 
Tracker I 
Tracker N I 
Volume I 
Storage server I I 
Storage server 12 
Storage server X 
Volume K 
Storage setNer K I 
Storage server K2 
Storage server KZ 

FastDFS的优点在于，跟踪器和存储结点的服务器可以随时增加或下线而不会影响服务。因为它的每个卷名(Group)存储的是同一组文件，这就起到了一个冗余备份和负载均衡的作用，而每个Group的容量是由一个Group中最小的那个服务器的容量决定的，这是FastDFS中特有的木板原理。那么如何增加整个分布式文件系统的容量呢? 可以通过增加Group来达到这个目的，FastDFS原生源码可以最多支持512个group，这是在tracker\_types.h头文件中定义的，如果实现二次开发，那么这个数字应该还可以更改。Group有一个最大长度为16位的名字定义(name)，而卷内的每个成员都有一个最大长度为16位的id定义。同时，因为拥有分卷的组织方式，所以FastFDs可以支持大容量的设计。一个FastDFS中所有卷的文件容量加起来就是存储系统的文件总容量。

文件名的标识：卷名+文件名

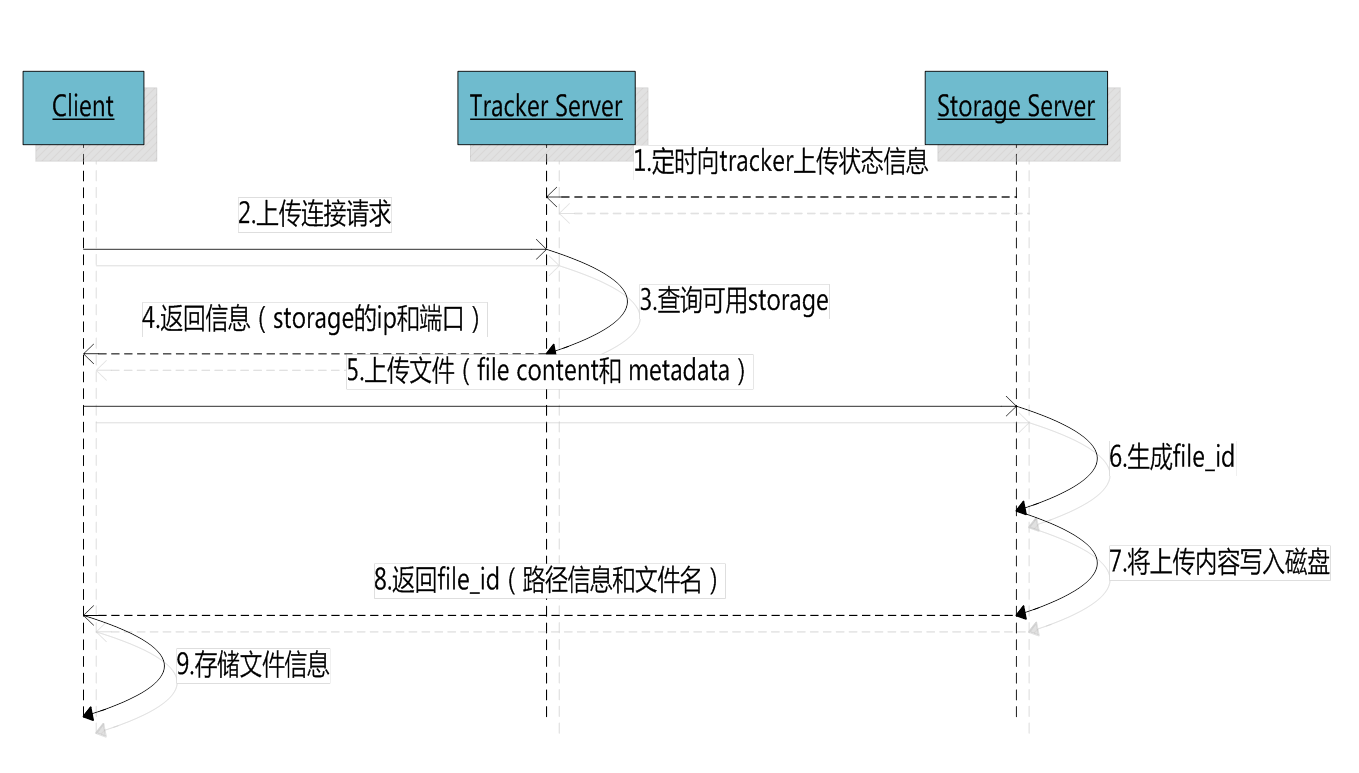
文件传输过程：Client 向Tracker请求文件，Tracker给予Client 客户文件标识和ip地址，Client再向Storage请求文件

上传文件交互过程：

1. client询问tracker上传到的storage，不需要附加参数；

2. tracker返回一台可用的storage；

3. client直接和storage通讯完成文件上传。

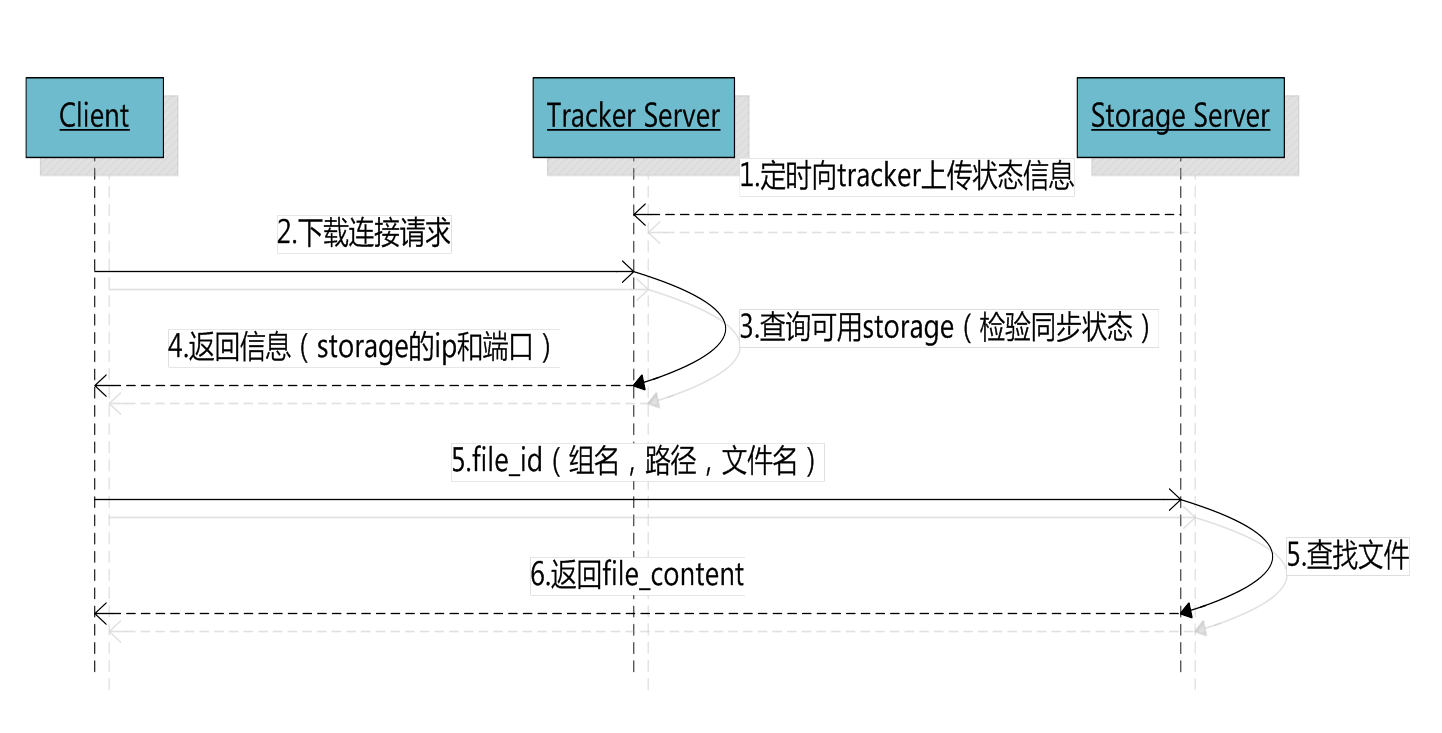


下载文件交互过程：

1. client询问tracker下载文件的storage，参数为文件标识（卷名和文件名）；

2. tracker返回一台可用的storage；

3. client直接和storage通讯完成文件下载



### 安装过程

1, 本次安装采用三台Ubuntu14.04LTS linux虚拟机

192.168.80.100 tracker Nginx(注意这台不安装fastsfd-niginx插件)

192.168.80.101 storage nginx

192.168.80.102 storage nginx

2, 准备编译环境 sudo apt-get install gcc gcc+ gcc-c++ openssl openssl-devel pcre pcre-devel

三台机器都进行安装，并且创建两个新用户fastdfs 和nginx

[y@ubuntu:~$]useradd fastdfs -M -s /sbin/nologin useradd nginx -M -s /sbin/nologin

3, 下载源码

敲 cd /usr/local/src/ 进入该目录下，运行如下命令，下载fastDFS 5.01

[y@ubuntu:~$]wget http://jaist.dl.sourceforge.net/project/fastdfs/FastDFS%20Server%20Source%20Code/FastDFS%20Server%20with%20PHP%20Extension%20Source%20Code%20V5.01/FastDFS\_v5.01.tar.

下载 nginx 1.7.0

[y@ubuntu:~$]wget http://nginx.org/download/nginx-1.7.0.tar.gz

下载fastdfs-nginx-module\_v1.16

[y@ubuntu:~$]#wget http://jaist.dl.sourceforge.net/project/fastdfs/FastDFS%20Nginx%20Module%20Source%20Code/fastdfs-nginx-module\_v1.16.tar.gz

4, 安装FastDFS （三台机器都要安装）

[y@ubuntu:~$]#tar xf FastDFS\_v5.01.tar.gz

[y@ubuntu:~$]#cd FastDFS

[y@ubuntu:~$]#./make.sh && ./make.sh install

5, 解压fastDFS-nginx-module

[y@ubuntu:~$]# cd /usr/local/src/

[y@ubuntu:/usr/local/src$]# tar xf fastdfs-nginx-module\_v1.16.tar.gz

6, 安装Nginx

192.168.80.100 tarcker 机器的安装

[y@ubuntu:~$]# cd /usr/local/src/

[y@ubuntu:/usr/local/src$]# tar xf nginx-1.7.0.tar.gz

[y@ubuntu:/usr/local/src$]# cd nginx-1.7.0

[y@ubuntu:/usr/local/src/nginx-1.7.0$]# ./configure --user=nginx --group=nginx --prefix=/usr/local/nginx

[y@ubuntu:/usr/local/src/nginx-1.7.0$]# make

[y@ubuntu:/usr/local/src/nginx-1.7.0$]# make install

192.168.80.101,102 stroage nginx的安装

[y@ubuntu:~$]# cd /usr/local/src/

[y@ubuntu:/usr/local/src$]# tar xf nginx-1.7.0.tar.gz

[y@ubuntu:/usr/local/src$]# cd nginx-1.7.0

[y@ubuntu:/usr/local/src/nginx-1.7.0$]# ./configure --user=nginx --group=nginx --prefix=/usr/local/nginx

--add-module=../fastdfs-nginx-module/src //storage 安装nginx时需要加载该模块

[y@ubuntu:/usr/local/src/nginx-1.7.0$]# make

[y@ubuntu:/usr/local/src/nginx-1.7.0$]# make install

三 配置

192.168.80.100 tracker的配置

1,创建tracker数据以及日志存放目录

[y@ubuntu:~$]# mkdir -p /data/fastdfs/tracker

2,修改FastDFS的tracker.conf配置 文件

[y@ubuntu:~$]# vim /etc/fdfs/tracker.conf

base\_path=/data/fastdfs/tracker

max\_connections=1024

work\_threads=8

store\_lookup=0

store\_path=0

reserved\_storage\_space=4G //

run\_by\_group=fastdfs

run\_by\_user=fastdfs

rotate\_error\_log=true

3,修改Nginx的配置文件

[y@ubuntu:~$]# vim /usr/local/nginx/conf/nginx.conf

user nginx nginx; //此处为已经建立好的用户 和分组

worker\_processes 3;

pid /usr/local/nginx/logs/nginx.pid;

worker\_rlimit\_nofile 1024;

events {

use epoll; // epoll是Linux内核为处理大批量文件描述符而作了改进的poll

worker\_connections 1024;

}

http {

include mime.types;

default\_type application/octet-stream;

log\_format main '$remote\_addr - $remote\_user [$time\_local] "$request" '

'$status $body\_bytes\_sent "$http\_referer" '

'"$http\_user\_agent" "$http\_x\_forwarded\_for"';

access\_log /usr/local/nginx/logs/access.log main;

upstream server\_g1{

server 192.168.80.101:80; //这里配置的是storage的IP 可以配多台

server 192.168.80.102:80;

}

server {

listen 80;

server\_name localhost;

location /g1 {

proxy\_redirect off;

proxy\_set\_header Host $host;

proxy\_set\_header X-Real-IP $remote\_addr;

proxy\_set\_header X-Forwarded-For $proxy\_add\_x\_forwarded\_for;

proxy\_pass http://server\_g1;

}

}

}

4,将tracker交给service管理并且设置开机启动

[y@ubuntu:~$]# cp /usr/local/src/FastDFS/init.d/fdfs\_trackerd /etc/init.d/

[y@ubuntu:~$]# chkconfig --add fdfs\_trackerd

[y@ubuntu:~$]# chkconfig fdfs\_trackerd on

配置storage （分别在192.168.80.101,102上进行配置）

1, 创建数据存放目录

[y@ubuntu:~$]# mkdir -p /data/fastdfs/storage/data

2,修改FastDFS的storage.conf配置文件

[y@ubuntu:~$]# vim /etc/fdfs/storage.conf

group\_name=g1

base\_path=/data/fastdfs

##工作线程数，通常设置为 CPU 数

work\_threads=8

store\_path\_count=1

store\_path0=/data/fastdfs/storage

##tracker\_server 的地址

tracker\_server=192.168.80.100:22122

##运行 FastDFS 的用户组

run\_by\_group=fastdfs

##运行 FastDFS 的用户

run\_by\_user=fastdfs

file\_distribute\_path\_mode=1

rotate\_error\_log=true

3,把nginx模块的配置文件拷贝到 /etc/fdfs中，进行修改

[y@ubuntu:~$]# cp /usr/local/src/fastdfs-nginx-module/src/mod\_fastdfs.conf /etc/fdfs/

[y@ubuntu:~$]# vim /etc/fdfs/mod\_fastdfs.conf

connect\_timeout=30

tracker\_server=192.168.80.100:22122

group\_name=g1

url\_have\_group\_name = true

store\_path\_count=1

store\_path0=/data/fastdfs/storage

4,修改nginx配置文件

[y@ubuntu:~$]# vim /usr/local/nginx/conf/nginx.conf

user nginx nginx;

worker\_processes 8;

pid /usr/local/nginx/logs/nginx.pid;

worker\_rlimit\_nofile 1024;

events {

use epoll;

worker\_connections 1024;

}

http {

include mime.types;

default\_type application/octet-stream;

log\_format main '$remote\_addr - $remote\_user [$time\_local] "$request" '

'$status $body\_bytes\_sent "$http\_referer" '

'"$http\_user\_agent" "$http\_x\_forwarded\_for"';

access\_log /usr/local/nginx/logs/access.log main;

server {

listen 80;

server\_name localhost;

location /g1/M00{

root /data/fastdfs/storage/data;

ngx\_fastdfs\_module;

}

}

}

5,把storage 交[y@ubuntu:~$]# cp /usr/local/src/FastDFS/init.d/fdfs\_storaged /etc/init.d/

[y@ubuntu:~$]# chkconfig --add fdfs\_storaged

[y@ubuntu:~$]# chkconfig fdfs\_storaged on

[y@ubuntu:~$]# service fdfs\_storaged start给service管理并设置开机启动

//创建软连接

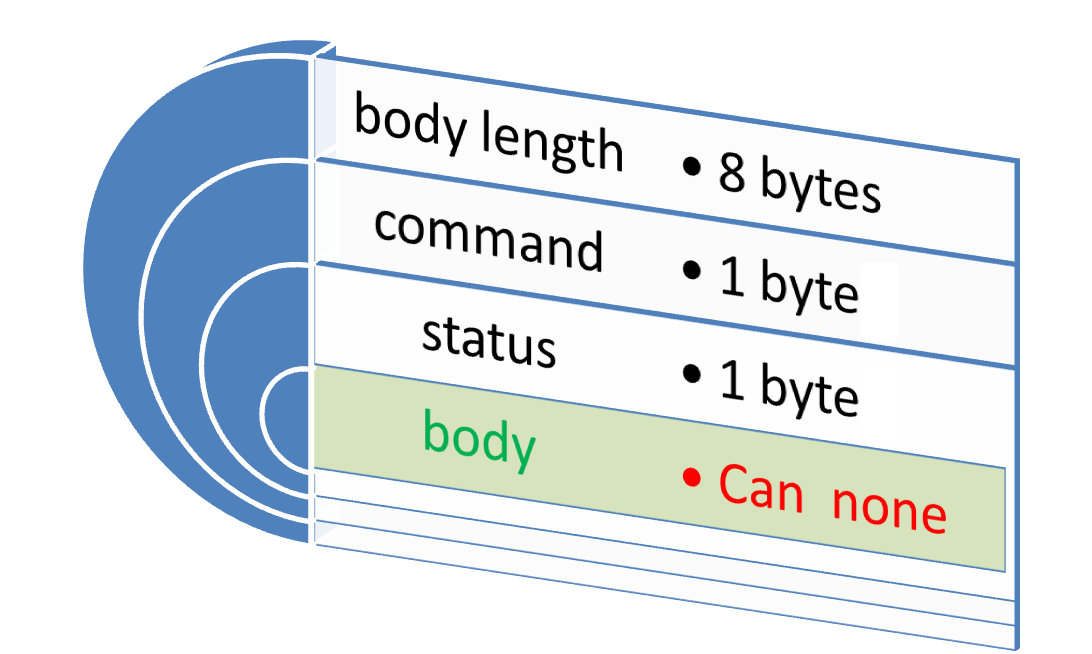
[y@ubuntu:~$]# ln -s /data/fastdfs/storage/data /data/fastdfs/storage/data/M00

以上，FastDFS就安装成功了

## Fastdfs分布式文件系统功能的具体细节

### 传输协议

FastDFS采用基于TCP/IP协议的方式来进行角色中的相互通信，它的协议报文如图所示



报文由两部分组成，一个是长度固定为10字节的header，包含了8字节的报文长度（不包含header长度）信息，1字节的报文命令信息和1字节的服务器状态响应码三种信息。所有协议定义都在tracker文件夹下的tracker\_proto.h头文件中。可以发现报文头没有包含固定格式的ip地址，其原因是因为根据命令的不同，需要通知的ip地址数目会有所区别。因为FastDFS是具有分组性质的，不管是group还是tracker，都可能会有一个或者多个成员。因此，像通知tracker组有新的tracker加入的消息的话，报文协议需要一次就通知到每个ip地址，这些ip地址和报文内容会按照固定顺序排列在body中。而且，并不是每一条命令都需要有ip地址的，因为集群中会保持一个长期的tcp协议socket，消息的传递只需要在socket中进行即可。

因为命令的长度有一个字节，所以潜在有256条角色之间的命令，而源码中内置的命令已经占用了72条。72条命令主要分为两大类，一类是tracker服务器发出的，针对storage和Storage队列映射表的管理的命令，一类是Storage发出的，针对组内文件同步，容量管理和垃圾空间回收的命令。这些命令构成了FastDFS角色之间的主要通信。剩余的命令空间还可以用来添加自定义的功能。

### Storage服务器数据存储

Storage服务器除了需要存储客户端上传的文件之外，还需要对已经存储的文件进行压缩和整合。对于一个上传到服务器的文件，只要成功被写入到Group中的一台机器中，FastDFS就视为文件上传成功，但是文件的备份需要Storage服务器自己进行。后台线程会写一份binlog文件，binlog里不包含文件数据，但是会包含文件名等元信息，记录每一次同步的时间，以便故障重启之后能继续该次同步。

### 文件的快速查找

文件名的命名方式在分布式文件系统中是一项及其精巧的设计，如果运用得到，就可以极大地减少文件位置查找对资源的消耗。在FastDFS中，文件索引(FID)包含如下的信息：卷名，虚拟磁盘路径，数据两级目录和文件名，具体构造如下图所示。



每一个文件索引都是在客户端上传文件之后，存储服务器返回给客户端的，这也是客户端以后访问文件需要用到的key值。虚拟磁盘是FastFDS的类似于分页的一种方式，可以减少数据损坏之后所耗费的修复时间，它的路径由Storage服务器自己配置，而数据两级目录是FastDFS为了避免单个目录下文件数目太多而内置的文件分层机制，最多可有256\*256，也就是65536个子目录，每个目录都可用来存储数据文件。值得注意的是，文件名是有FastDFS的Storage服务器根据文件具体的信息生成的标志，是与客户端直接上传的文件名称截然不同的。

文件名的生成是由一个int型的源group ip地址，一个time\_t类型的时间戳(文件创建的时间)，一个int64\_t类型的文件大小（如果文件大小小于32位则包含随机数），一个int型的crc32文件校验值拼接而成，然后对这个二进制串进行base64编码，转为可打印的字符串而成，文件名的后缀(如上文的.sh)是由客户端指定的，可以用来区分文件类型。通过这种方式就可以拼凑一个反应文件信息又相互独立不会重名的文件了。

### Master单点

心跳机制和对等性。

## 性能测试

## 随便什么的大杂烩

啊啊啊啊啊啊啊啊啊啊啊啊啊啊啊啊啊啊啊啊啊啊啊啊啊啊啊啊啊啊啊啊啊啊啊啊啊啊啊啊啊啊啊啊啊啊啊啊啊啊啊啊啊啊啊啊啊啊啊啊啊啊啊啊啊啊啊啊啊啊啊啊啊啊啊啊啊啊啊

我们通常对于一个数据存储系统有如下要求有1、数据容量和负载均衡2、系统容错能力3、系统扩展能力。容量和负载均衡是分布式系统相对于传统单机文件系统的天然的优势，然而FastDFS是如何实现系统的容错性和扩展性呢？前面我们提到了，FastDFS不会因为tracker的增加或减少而影响性能，所以这样的对等性就是它优秀的扩展性的良好体现。那么它的内部协议是如何做到容错性的呢？storage 服务器主要实现三个功能，接受和处理来自client或其他storage的请求，将自己的信息报告给tracker server 和同步本地数据到同组其它的storage。当storage 服务器向tracker 服务器行驶报告信息这个功能时，如果有新的tracker加入，storage连接之后就会发现该tracker返回的storage server列表比本机要少，于是storage server会把tracker上没有的列表传递给它。那么storage server是如何发现新的tracker server的呢？ 是因为，这不是一个像AD-hoc一样的自组织协议，当需要增加新的tracker server的时候，只需要在每个storage的配置文件里新增tracker server的端口号和ip地址即可。Tracker server之间是否不相互通信呢？由于tracker server之间是完全对等的，所以tracker集群中的所有服务器都是靠storage的报告建立目录映射的，因此tracker集群中的主机不需要相互通信。无论集群怎么扩和维护，client是不受影响的。这点可以多研究一下fastdfs的client api就会明白？？？虽然只有storage向tracker报告自己的状态，但是tracker server会定时检查每个group的心跳来，（将心跳时间过期的storage设置为offline。那如果客户请求的文件的所有storage都offline了怎么办呢？）如果一个group中有服务器出现故障的话，那就涉及到单盘故障恢复问题了。

单盘故障恢复，是指一个group中有服务器磁盘出现故障了的情况，只要group中还有一台服务器是运行正常的，那么它的磁盘中就有该group文件的所有备份，因此FastDFS可以原生支持将旧磁盘的数据拷贝到新硬盘的过程，自动检测完成。当一个新的磁盘挂载到服务器上时，如果FastDFS的Storage程序发现这是非首次启动而磁盘里没有任何数据，就会开始内部恢复过程。恢复分为三个步骤：RecoveryStart、RecoveryStore和RecoveryFinish。一旦恢复过程开始，RecoveryStore就会记录中间状态（？用什么记录），因此即使磁盘在恢复的过程中宕机，下次启动还是可以继续进行上次恢复的步骤。ReoveryStore的状态文件会在RecoveryStart开始时初始化，它们是.recovery.mark和.binlog.recovery文件。.binlog.recovery会记录现在正在下载的文件名，而.recovert.mark会记录此时的存储状态，下载的binlog的偏移量和binlog是否下载完成。（什么是binlog呢）。RecoveryStart还会向Tracker报告自己此时的状态，从Tracker中找到最适合同步的文件，每1000个文件记录binlog offset的值。如果全部下载完成，则RecoveryFinish函数负责将临时文件删除。但是磁盘修复的速度取决于磁盘写入的速度，如果是4TB的盘，100MB/s的写入速度，一块磁盘的恢复至少需要11个小时，而且下载过程是单线程同步的方式进行的，受限于网络IO和磁盘读入速度，以及下载时会占用一个storage服务器的网络资源，并且由于是同步方式，进行磁盘恢复中的Storage服务器无法处理服务上的事情。

单盘故障恢复过程和新增Storage服务器同步过程有十分相似的地方，不过一个是新增，一个是恢复。添加机器的同步叫做源同步。源同步是把所有数据都同步到新机器上。如果是客户端上传了文件因而造成group内数据不一致而发生的同步，FastDFS会采用增量同步的方式。Storage首次启动时，会向Tracker发送一个同步请求，如果组内有可以同步的机器，Tracker会向Storage指出应该将哪台机器作为源服务器，并且获得源机器ip地址和旧数据截止时间。如果组内没有其它机器，Tracker会向Storage服务器指出不需要同步。如果组内有机器但没有适合同步的机器，Tracker就会向Storage返回一个错误，Storage会根据这个错误睡眠一段时间在发出同步请求。（机器状态？）。获取到ip地址和截止时间之后，新Storage会向Tracker发送心跳信息，等到同步完成，新Storage会再向Tracker发送一个心跳信息，此时新增服务器同步完成。

FastDFS会把集群中的所有机器设置几个状态，目前已知的是ONLINE，OFF\_LINE，WAIT\_SYNC，ACTIVE，RECOVERY。这些状态代表的含义从名字上看是很清晰的，只不过这些状态的数据结构是什么，储存在什么地方，都是需要探究的。还需要了解的就是binlog文件和trunk合并。

Binlog文件是FastDFS系统为了进行一个group中增量文件备份而设置的特别的记录文件。在配置文件Storage.conf设置的base\_path路径下，会在程序首次启动时创建一个base\_path/data/sync目录，目录下就会存放相关同步文件，同时和磁盘恢复过程一样，增量同步也会有一个.mark文件来记录文件同步的状态。在.mark文件中，会有两项和binlog文件有关，binlog\_index和binlog\_offset，前者表示上次同步的最后一条文件索引，后者表示同步到该文件位置的偏移量。如果出现了重启的情况，同步只要从该记录的位置继续进行即可。Binlog文件内部是一条一条的日志记录，格式是 时间戳 创建文件的方式 和文件名fileid，创建文件的方式分为创建C，追加A，删除D和删减T。大写表示源操作，小写表示副本操作。（源操作是指同步给其它Storage的文件，而副本就是同步到其它Storage的文件），文件名fileid的精巧构造已经探索过了，名字中反应了文件的元数据信息，一定程度上减少了IO的压力。同步过程如下 1、启动时创建一个线程来保持和Tracker的通信 2、每隔30秒发送心跳包，并得到Tracker的返回报文，其中包含其他Storage服务器的信息 3、为每个同group内的Storage服务器创建一个线程进行同步 4、每个对应Storage的线程打开对应ip地址的.mark文件，找到需要同步文件的位置和偏移量，进入一个while循环以阻塞的方式同步给对方。