6-产品上云和性能测试

零声教育 Darren QQ: 326873713
https://ke.qq.com/course/420945?tuin=137bb271

本节内容

- 小文件存储原理
- 性能测试
- 优化策略
- 项目问题

网页版本：https://www.yuque.com/docs/share/bca5e11a-a981-479c-9ab4-0cb71a5d8261?#
《6-产品上云和性能测试》

《fastdfs-扩展阅读资料.rar》对应的文档，大家带着思辨的心态去阅读，里面的观点不一定都对。

1 海量小文件存储

重点内容

- 什么是海量小文件存储，多小为小，多少文件为海量
- 为什么要针对海量小文件存储做优化
- 小文件如何存储-存储结构
- 小文件如何查找-查找的条件
- 小文件如何删除-删除后文件是否需要移动，如果文件不移动，会不会造成碎片空间
- 小文件如何更新-能否更新

1.1 为什么需要小文件存储

1.1.1 小文件应用场景

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，以下简称LOSF。

LOSF应用在目前实际中越来越常见，如社交网站、电子商务、广电、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小图片定制优化的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，也推出了针对小文件优化的TFS文件系统存储这些图片，并且进行了开源。
- 歌华有线可以进行图书和视频的在线点播，图书每页会扫描成一个几十KB大小的图片，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流切割成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、纹理等原理素材，一部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，单个文件大小为几KB至几百KB的不等，文件数量达到数千万乃至数亿，并且逐年增长。

1.1.2 小文件存储带来的问题

Linux通过node存储文件信息，但inode也会消耗硬盘空间，所以硬盘格式化的时候，操作系统自动将硬盘分成两个区域。

1. 一个是数据区，存放文件数据；
2. 另一个是inode区（inode table），存放inode所包含的信息。

每个inode节点的大小，一般是128字节或256字节。inode节点的总数，在格式化时就给定，一般是每1KB或每2KB就设置一个inode。假定在一块1GB的硬盘中，每个inode节点的大小为128字节，每1KB就设置一个inode，那么inode table的大小就会达到128MB，占整块硬盘的12.8%。

小文件主要有2个问题：

1. 如果小文件都小于<1KB，而系统初始化的时候每个2K设置一个node，则此时一个文件还是至少占用2K的空间，最后导致磁盘空间利用率不高，< 50%。**点击引申阅读**
2. 大量的小文件，导致在增加、查找、删除文件的时候需要遍历过多的node节点，影响效率。**点击引申阅读**

1.2 小文件机制配置

合并文件存储相关的配置都在**tracker.conf**中。配置完成后，重启tracker和storage server。支持小文件存储，**只需要设置tracker的use_trunk_file=true，** store_server=1， **其他保持默认即可，但也要注意**slot_max_size 的大小，这样才知道多大的文件触发小文件存储机制**。**

#是否启用trunk存储，缺省false，需要打开配置
use_trunk_file = true

#trunk文件最小分配单元 字节，缺省256，即使上传的文件只有10字节，也会分配这么多空间。
slot_min_size = 256

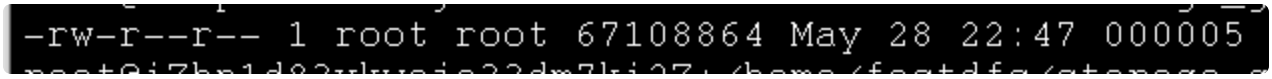
#trunk内部存储的最大文件，超过该值会被独立存储，缺省16M，超过这个size的文件，不会存储到trunk file中，而是作为一个单独的文件直接存储到文件系统中。
slot_max_size = 1MB

# 文件 trunk空间对齐
# default value is 0 (never align)
# NOTE: the larger the alignment size, the less likely of disk
#   fragmentation, but the more space is wasted.
trunk_alloc_alignment_size = 256

#连续空虚的空间是否合并　　256　512 = 768合并后
# default value is false
trunk_free_space_merge = true

# if delete / reclaim the unused trunk files
# default value is false
delete_unused_trunk_files = false

#trunk文件大小，默认 64MB，不要配置得过大或者过小，最好不要超过256MB。
trunk_file_size = 64MB



#是否预先创建trunk文件，缺省false
trunk_create_file_advance = false

#预先创建trunk文件的基准时间
trunk_create_file_time_base = 02:00

#预先创建trunk文件的时间间隔 一天
trunk_create_file_interval = 86400

the threshold to create trunk file
when the free trunk file size less than the threshold,
will create he trunk files
#trunk创建文件的最大空闲空间，如果空闲的trunk file空间大于本参数，则不会提前创建。
trunk_create_file_space_threshold = 20G

if check trunk space occupying when loading trunk free spaces
the occupied spaces will be ignored
default value is false
since V3.09
NOTICE: set this parameter to true will slow the loading of trunk spaces
when startup. you should set this parameter to true when neccessary.
#启动时是否检查每个空闲空间列表项已经被使用
trunk_init_check_occupying = false

if ignore storage_trunk.dat, reload from trunk binlog
default value is false
set to true once for version upgrade when your version less than V3.10
#是否纯粹从trunk-binlog重建空闲空间列表
trunk_init_reload_from_binlog = false

the min interval for compressing the trunk binlog file
unit: second, 0 means never compress
FastDFS compress the trunk binlog when trunk init and trunk destroy
recommand to set this parameter to 86400 (one day)

```
# default value is 0
trunk_compress_binlog_min_interval = 86400

# the interval for compressing the trunk binlog file
# unit: second, 0 means never compress
# recommand to set this parameter to 86400 (one day)
# default value is 0
#对trunk-binlog进行压缩的时间间隔
trunk_compress_binlog_interval = 86400

# compress the trunk binlog time base, time format: Hour:Minute
# Hour from 0 to 23, Minute from 0 to 59
# default value is 03:00
trunk_compress_binlog_time_base = 03:00
```

1.3 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。

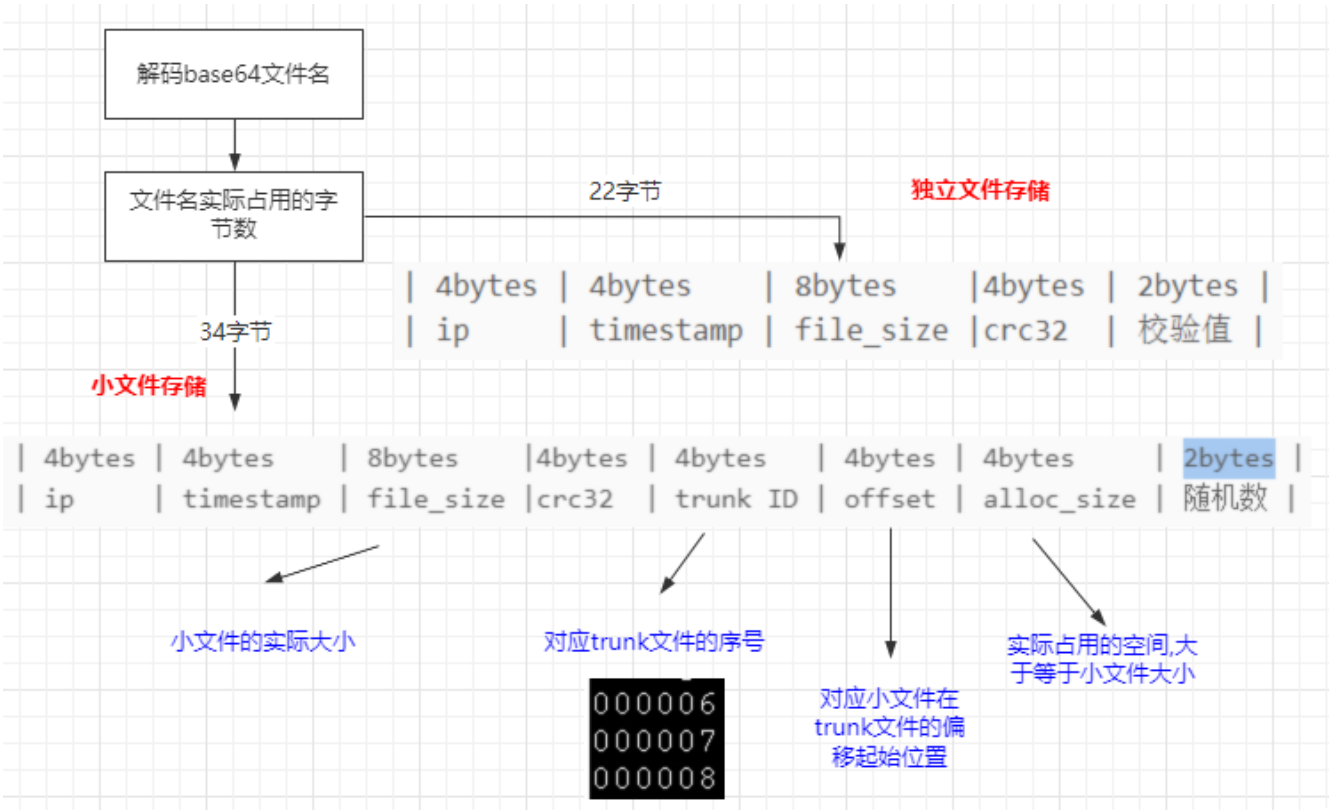
注：下面将采用合并存储后的大文件统称为**Trunk文件**，没有合并存储的文件统称为**源文件**；

请大家注意区分三个概念：

1）Trunk文件：storage服务器磁盘上存储的实际文件，默认大小为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文件文件名格式：fdfs_storage1/data/00/00/**000001** 文件名从1开始递增，类型为int；

1.3.1 在启动小文件存储时服务返回给客户端的fileid有变化



1. 独立文件存储的file id

文件名（不含后缀名）采用Base64编码，包含如下5个字段（每个字段均为4字节整数）：

> group1/M00/00/00/wKgqHV4OQQyAbo9YAAAA_fdSpmg855.txt

这个文件名中，除了.txt为文件后缀，wKgqHV4OQQyAbo9YAAAA_fdSpmg855 这部分是一个base64编码缓冲区，组成如下：

- storage_id（ip的数值型）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（文件内容的检验码）
- 随机数 （引入随机数的目的是防止生成重名文件）

Bash 复制代码

```
wKgqHV4OQQyAbo9YAAAA_fdSpmg855
| 4bytes | 4bytes | 8bytes | 4bytes | 2bytes |
| ip | timestamp | file_size | crc32 | 随机数 |
```

2 小文件存储的file id

如果采用了合并存储，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，包含如下几个字段：

> group1/M00/00/00/eBuDxWCwrDqITi98AAAA-3Qtcs8AAAAAQAAAgAAAAIA833.txt

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：

- storage_id（ip的数值型）源storage server ID或IP地址
- timestamp（文件创建时间戳）
- file_size：实际的文件大小
- crc32：文件内容的crc32码
- trunk file ID：大文件ID如000001
- offset：文件内容在**trunk文件中的偏移量**
- alloc_size：分配空间，大于或等于文件大小
- 随机数 （引入随机数的目的是防止生成重名文件

```
eBuDxWCwrDqITi98AAAA-3Qtcs8AAAAAQAAAgAAAAIA833
| 4bytes | 4bytes    | 8bytes    |4bytes | 4bytes   | 4bytes | 4bytes
| 2bytes |
| ip     | timestamp | file_size |crc32  | trunk ID | offset | alloc_size
| 随机数 |
```

1.3.2 Trunk文件存储结构

1.3.2.1 磁盘数据内部结构

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，结构如下：

```
|||————————————————————— 24bytes
————————————————————————————————————|||
|–1byte   –|–  4bytes  –|– 4bytes –|–4bytes– |–4bytes–|————    7bytes
 ————|
|–filetype–|–alloc_size–|–filesize–|–crc32  –|–mtime –|–
formatted_ext_name–|
|||————————————————————— file_data filesize bytes
————————————————————|||
|————————————————————— file_data
————————————————————————————————————————|
```

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的文件时，也会产生空余空间。

即是：每次创建一个trunk文件时，是安装既定的文件大小去创建该文件trunk_file_size，当要删除trunk文件时，需要该trunk文件没有存储小文件才能删除。

1.3.2.2小文件存储平衡树

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。每当需要存储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：

- 要求存储文件为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300KB返回用于存储，后面50KB则继续放置到空闲平衡树之中。
- 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（此时总是能够满足了）。
- **进一步阅读：storage\trunk_mgr\trunk_mem.c 搜索** avl_tree_insert、avl_tree_find等函数。

1.3.3 调用trunk_create_trunk_file创建trunk文件

trunk_create_trunk_file 创建trunk文件

#0 __libc_open64 (file=file@entry=0x7f098537d558
"/home/fastdfs/storage_group1_23000/data/00/00/000001", oflag=2)
    at ../sysdeps/unix/sysv/linux/open64.c:36
#1 0x0000559d07b346ee in open64 (__oflag=<optimized out>,
    __path=0x7f098537d558 "/home/fastdfs/storage_group1_23000/data/00/00/000001")
    at /usr/include/x86_64-linux-gnu/bits/fcntl2.h:59
#2 dio_open_file (pFileContext=pFileContext@entry=0x7f098537d558) at storage_dio.c:261
#3 0x0000559d07b35295 in dio_check_trunk_file_when_upload (pTask=<optimized out>)
    at storage_dio.c:786
#4 0x0000559d07b34a76 in dio_write_file (pTask=0x7f098537d4a8) at storage_dio.c:421
#5 0x0000559d07b34091 in dio_thread_entrance (arg=0x559d09463698) at storage_dio.c:748
#6 0x00007f0989f746db in start_thread (arg=0x7f0985138700) at pthread_create.c:463
#7 0x00007f0989a4a71f in clone () at ../sysdeps/unix/sysv/linux/x86_64/clone.S:95

#0 __libc_write (fd=fd@entry=4, buf=buf@entry=0x559d0943ede0, nbytes=nbytes@entry=100)
at ../sysdeps/unix/sysv/linux/write.c:27
#1 0x00007f0989d2dab8 in fc_safe_write (fd=4,
    buf=buf@entry=0x559d0943ede0 "1622207507 D 0 0 0 1 1536 67107328\n1622207507 A 0 0 0
1 2048 67106816\n1622207507 A 0 0 0 1 1536 512\n", nbyte=100) at shared_func.c:2705
#2 0x0000559d07b3ebf8 in trunk_binlog_fsync_ex (bNeedLock=bNeedLock@entry=true,
    buff=0x559d0943ede0 "1622207507 D 0 0 0 1 1536 67107328\n1622207507 A 0 0 0 1 2048
67106816\n1622207507 A 0 0 0 1 1536 512\n", length=length@entry=0x559d07d70910
<trunk_binlog_write_cache_len>)
    at trunk_mgr/trunk_sync.c:1274
#3 0x0000559d07b3fd58 in trunk_binlog_sync_func (args=<optimized out>)
    at trunk_mgr/trunk_sync.c:401
#4 0x00007f0989d3e3c3 in sched_thread_entrance (args=0x559d094631d0) at
sched_thread.c:469
#5 0x00007f0989f746db in start_thread (arg=0x7f098523a700) at pthread_create.c:463
#6 0x00007f0989a4a71f in clone () at ../sysdeps/unix/sysv/linux/x86_64/clone.S:95

#0 __libc_write (fd=fd@entry=23, buf=buf@entry=0x7f098537d959, nbytes=nbytes@entry=251)
 at ../sysdeps/unix/sysv/linux/write.c:27
#1 0x00007f0989d2dab8 in fc_safe_write (fd=23,
    buf=buf@entry=0x7f098537d959
"https://github.com/happyfish100/fastdfs/archive/V6.07.tar.gz\nhttps://github.com/happyfish

100/fastdfs-nginx-module/archive/V1.22.tar.gz\nhttps://github.com/happyfish100/libfastcommon/archive/V1.0.50.tar"..., nbyte=nbyte@entry=251) at shared_func.c:2705
#2 0x0000559d07b34ac1 in dio_write_file (pTask=0x7f098537d4a8) at storage_dio.c:437
#3 0x0000559d07b34091 in dio_thread_entrance (arg=0x559d09463698) at storage_dio.c:748
#4 0x00007f0989f746db in start_thread (arg=0x7f0985138700) at pthread_create.c:463
#5 0x00007f0989a4a71f in clone () at ../sysdeps/unix/sysv/linux/x86_64/clone.S:95

Thread 9 "fdfs_storaged" hit Breakpoint 2, __libc_write (fd=fd@entry=23, buf=buf@entry=0x7f0985137e60, nbytes=nbytes@entry=24) at ../sysdeps/unix/sysv/linux/write.c:27
27 in ../sysdeps/unix/sysv/linux/write.c
(gdb) bt
#0 __libc_write (fd=fd@entry=23, buf=buf@entry=0x7f0985137e60, nbytes=nbytes@entry=24)
    at ../sysdeps/unix/sysv/linux/write.c:27
#1 0x00007f0989d2dab8 in fc_safe_write (fd=23, buf=buf@entry=0x7f0985137e60 "F",
    nbyte=nbyte@entry=24) at shared_func.c:2705
#2 0x0000559d07b35454 in dio_write_chunk_header (pTask=<optimized out>) at storage_dio.c:915
#3 0x0000559d07b34c06 in dio_write_file (pTask=0x7f098537d4a8) at storage_dio.c:515
#4 0x0000559d07b34091 in dio_thread_entrance (arg=0x559d09463698) at storage_dio.c:748
#5 0x00007f0989f746db in start_thread (arg=0x7f0985138700) at pthread_create.c:463
#6 0x00007f0989a4a71f in clone () at ../sysdeps/unix/sysv/linux/x86_64/clone.S:95

#0 __libc_write (fd=fd@entry=5, buf=buf@entry=0x7f0985239640, nbytes=nbytes@entry=1076)
 at ../sysdeps/unix/sysv/linux/write.c:27
#1 0x00007f0989d2dab8 in fc_safe_write (fd=fd@entry=5,
    buf=buf@entry=0x7f0985239640
"total_upload_count=61\nsuccess_upload_count=61\ntotal_append_count=0\nsuccess_append_count=0\ntotal_modify_count=0\nsuccess_modify_count=0\ntotal_truncate_count=0\nsuccess_truncate_count=0\ntotal_download_cou"..., nbyte=nbyte@entry=1076) at shared_func.c:2705
#2 0x0000559d07b1b282 in storage_write_to_fd (fd=5,
    filename_func=filename_func@entry=0x559d07b1afc0 <get_storage_stat_filename>,
    pArg=pArg@entry=0x0,
    buff=buff@entry=0x7f0985239640
"total_upload_count=61\nsuccess_upload_count=61\ntotal_append_count=0\nsuccess_append_count=0\ntotal_modify_count=0\nsuccess_modify_count=0\ntotal_truncate_count=0\nsuccess_truncate_count=0\ntotal_download_cou"..., len=len@entry=1076) at storage_func.c:315
#3 0x0000559d07b1b62d in storage_write_to_stat_file () at storage_func.c:627

```
#4  0x0000559d07b2cbca in fdfs_stat_file_sync_func (args=<optimized out>) at
storage_service.c:8291
#5  0x00007f0989d3e3c3 in sched_thread_entrance (args=0x559d094631d0) at
sched_thread.c:469
#6  0x00007f0989f746db in start_thread (arg=0x7f098523a700) at pthread_create.c:463
#7  0x00007f0989a4a71f in clone () at ../sysdeps/unix/sysv/linux/x86_64/clone.S:95

Thread 9 "fdfs_storaged" hit Breakpoint 2, __libc_write (fd=16, buf=buf@entry=0x7f0985137e60,
    nbytes=nbytes@entry=8) at ../sysdeps/unix/sysv/linux/write.c:27
27   in ../sysdeps/unix/sysv/linux/write.c
(gdb) bt
#0  __libc_write (fd=16, buf=buf@entry=0x7f0985137e60, nbytes=nbytes@entry=8)
    at ../sysdeps/unix/sysv/linux/write.c:27
#1  0x0000559d07b1fe2f in storage_nio_notify (pTask=pTask@entry=0x7f098537d4a8)
    at storage_service.c:1927
#2  0x0000559d07b2bf05 in storage_upload_file_done_callback (pTask=0x7f098537d4a8,
    err_no=<optimized out>) at storage_service.c:1219
#3  0x0000559d07b34c30 in dio_write_file (pTask=0x7f098537d4a8) at storage_dio.c:525
#4  0x0000559d07b34091 in dio_thread_entrance (arg=0x559d09463698) at storage_dio.c:748
#5  0x00007f0989f746db in start_thread (arg=0x7f0985138700) at pthread_create.c:463
#6  0x00007f0989a4a71f in clone () at ../sysdeps/unix/sysv/linux/x86_64/clone.S:95

Thread 7 "sched" hit Breakpoint 2, __libc_write (fd=fd@entry=7,
buf=buf@entry=0x559d0944edf0,     nbytes=nbytes@entry=74) at
../sysdeps/unix/sysv/linux/write.c:27
27   in ../sysdeps/unix/sysv/linux/write.c
(gdb) bt
#0  __libc_write (fd=fd@entry=7, buf=buf@entry=0x559d0944edf0, nbytes=nbytes@entry=74)
    at ../sysdeps/unix/sysv/linux/write.c:27
#1  0x00007f0989d2dab8 in fc_safe_write (fd=7,
    buf=0x559d0944edf0 "1622207507 C M00/00/00/eBuDxWCw7BOIMeN8AAAA-
3Qtcs8AAAAAQAABgAAAAIA210.txt\n", nbyte=nbyte@entry=74) at shared_func.c:2705
#2  0x0000559d07b2e024 in storage_binlog_fsync (bNeedLock=bNeedLock@entry=true)
    at storage_sync.c:1601
#3  0x0000559d07b30a9c in fdfs_binlog_sync_func (args=<optimized out>) at
storage_sync.c:1576
#4  0x00007f0989d3e3c3 in sched_thread_entrance (args=0x559d094631d0) at
sched_thread.c:469
#5  0x00007f0989f746db in start_thread (arg=0x7f098523a700) at pthread_create.c:463
#6  0x00007f0989a4a71f in clone () at ../sysdeps/unix/sysv/linux/x86_64/clone.S:95
```

1.3.3 上传下载trunk文件

对于小文件存储，我们主要关注原理，具体的源码实现有兴趣的朋友可以阅读对应的源码。

1.3.3.1 trunk上传文件

#0 blocked_queue_push (pQueue=0x559d09463698, pTask=pTask@entry=0x7f098537d4a8) at fast_blocked_queue.c:62
#1 0x0000559d07b344fe in storage_dio_queue_push (pTask=pTask@entry=0x7f098537d4a8) at storage_dio.c:158
#2 0x0000559d07b213ea in storage_write_to_file (pTask=pTask@entry=0x7f098537d4a8, file_offset=file_offset@entry=2072, upload_bytes=upload_bytes@entry=251, buff_offset=<optimized out>, deal_func=<optimized out>, done_callback=done_callback@entry=0x559d07b2bda4 <storage_upload_file_done_callback>, clean_func=0x559d07b35081 <dio_trunk_write_finish_clean_up>, store_path_index=0) at storage_service.c:7459

```
4658
4659    if (pFileContext->extra_info.upload.file_type & _FILE_TYPE_TRUNK)       // trunk文件
```

小文件存储不是由客户端决定，是服务器去配置，具体由tracker.conf配置。

#3 0x0000559d07b2a00d in storage_upload_file (pTask=pTask@entry=0x7f098537d4a8, bAppenderFile=bAppenderFile@entry=false) at storage_service.c:4729
#4 0x0000559d07b2cd26 in storage_deal_task (pTask=pTask@entry=0x7f098537d4a8) 和普通上传文件一个逻辑
at storage_service.c:8345
#5 0x0000559d07b33b79 in client_sock_read (sock=22, event=<optimized out>, arg=0x7f098537d4a8)
at storage_nio.c:409
#6 0x00007f0989d43d34 in deal_ioevents (ioevent=0x559d09452fd0) at ioevent_loop.c:32
#7 ioevent_loop (pThreadData=pThreadData@entry=0x559d09452fd0, recv_notify_callback=<optimized out>, clean_up_callback=0x559d07b337ad <task_finish_clean_up>, continue_flag=0x559d07d6f368 <g_continue_flag>) at ioevent_loop.c:129
#8 0x0000559d07b1f98a in work_thread_entrance (arg=0x559d09452fd0) at storage_service.c:1960
#9 0x00007f0989f746db in start_thread (arg=0x7f098a260700) at pthread_create.c:463
#10 0x00007f0989a4a71f in clone () at ../sysdeps/unix/sysv/linux/x86_64/clone.S:95

storage_upload_file

```
if (pFileContext->extra_info.upload.file_type & _FILE_TYPE_TRUNK)       // 小文件   {
        FDFSTrunkFullInfo *pTrunkInfo;

        pFileContext->extra_info.upload.if_sub_path_alloced = true;
        pTrunkInfo = &(pFileContext->extra_info.upload.trunk_info);
        if ((result=trunk_client_trunk_alloc_space( \
            TRUNK_CALC_SIZE(file_bytes), pTrunkInfo)) != 0)
        {
            return result;
        }

        clean_func = dio_trunk_write_finish_clean_up;
        file_offset = TRUNK_FILE_START_OFFSET((*pTrunkInfo));
   pFileContext->extra_info.upload.if_gen_filename = true;
        trunk_get_full_filename(pTrunkInfo, pFileContext->filename, \
                sizeof(pFileContext->filename));
        pFileContext->extra_info.upload.before_open_callback = \
                    dio_check_trunk_file_when_upload;
        pFileContext->extra_info.upload.before_close_callback = \
                    dio_write_chunk_header;
        pFileContext->open_flags = O_RDWR | g_extra_open_file_flags;
    }
```

1.3.3.2 trunk下载文件

```
#0  blocked_queue_push (pQueue=0x559d09463630, pTask=pTask@entry=0x7f098537d4a8)   at
fast_blocked_queue.c:62
#1  0x0000559d07b344fe in storage_dio_queue_push (pTask=pTask@entry=0x7f098537d4a8)
   at storage_dio.c:158
#2  0x0000559d07b1ff67 in storage_read_from_file (pTask=pTask@entry=0x7f098537d4a8,
   file_offset=file_offset@entry=2072, download_bytes=download_bytes@entry=251,
   done_callback=done_callback@entry=0x559d07b238af
<storage_download_file_done_callback>,
   store_path_index=<optimized out>) at storage_service.c:7405
#3  0x0000559d07b201d6 in storage_server_download_file
(pTask=pTask@entry=0x7f098537d4a8)
   at storage_service.c:7336
#4  0x0000559d07b2cc3f in storage_deal_task (pTask=pTask@entry=0x7f098537d4a8)
   at storage_service.c:8331
```

#5 0x0000559d07b33b79 in client_sock_read (sock=23, event=<optimized out>, arg=0x7f098537d4a8)
    at storage_nio.c:409
#6 0x00007f0989d43d34 in deal_ioevents (ioevent=0x559d094530b8) at ioevent_loop.c:32
#7 ioevent_loop (pThreadData=pThreadData@entry=0x559d094530b8,
    recv_notify_callback=<optimized out>, clean_up_callback=0x559d07b337ad <task_finish_clean_up>,
    continue_flag=0x559d07d6f368 <g_continue_flag>) at ioevent_loop.c:129
#8 0x0000559d07b1f98a in work_thread_entrance (arg=0x559d094530b8) at storage_service.c:1960
#9 0x00007f0989f746db in start_thread (arg=0x7f098533c700) at pthread_create.c:463
#10 0x00007f0989a4a71f in clone () at ../sysdeps/unix/sysv/linux/x86_64/clone.S:95

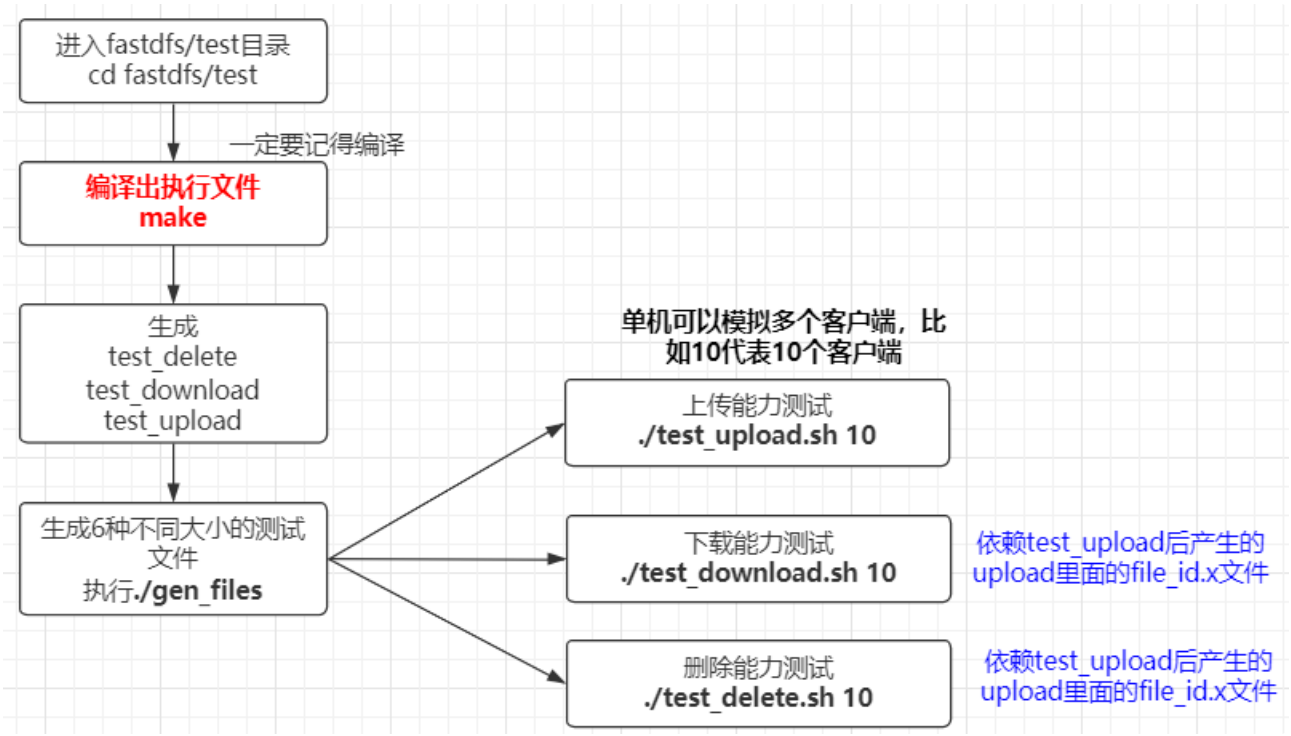
2 只允许唯一文件上传

storage.conf 配置
# if check file duplicate, when set to true, use FastDHT to store file indexes# 1 or yes: need check
# 0 or no: do not check
# default value is 0
check_file_duplicate = 0
这里依赖FastDHT去处理，本质而言就是对上传的文件内容进行hash计算或者md5计算出来唯一值，并保存，然后每次上传文件统计出来的唯一值都去存储的数据里面查找，如果有同样的则认为文件重复。

但这种方式导致每个上传文件都要在服务器做实时唯一值计算，比较消耗cpu的资源，建议还是类似我们图床的方式去做：

- 文件MD5唯一值由客户端计算并发送给文件服务器;
- 文件服务器得到客户端提交的MD5值后，到redis+mysql的组合里面查询是否存在同样的文件。

3 性能测试



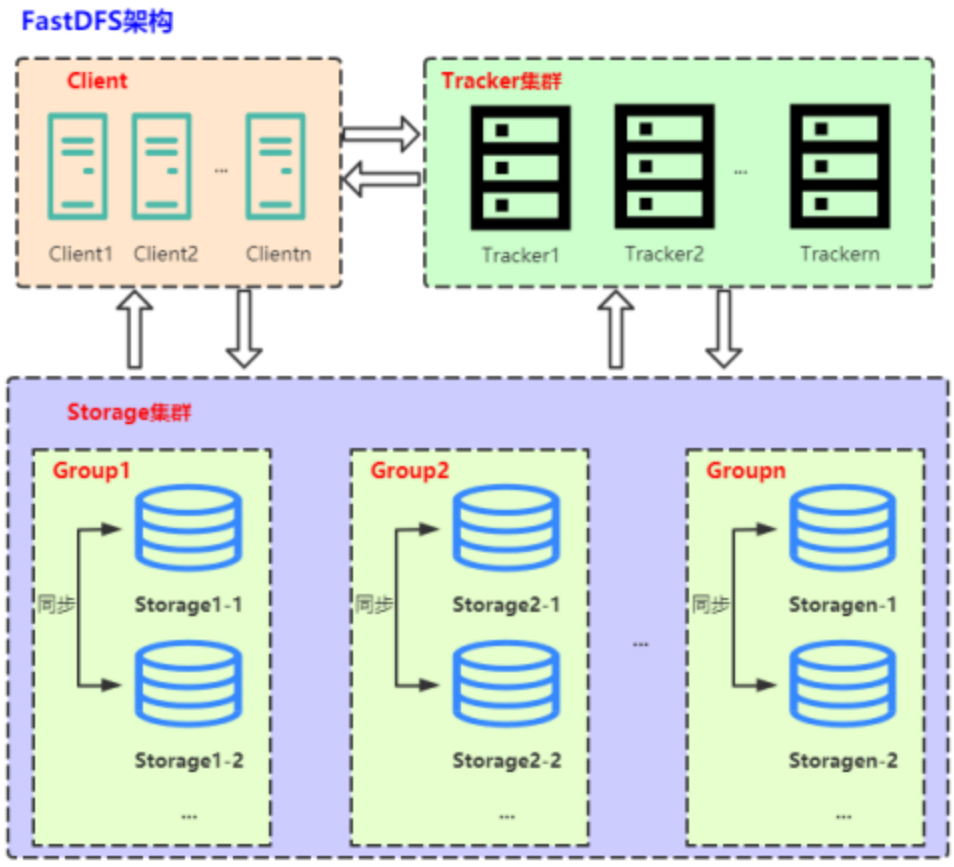
小规模测试的时候，建议一台客户端机器只模拟一个客户端 ./test_upload.sh 1 。

可以提前看看fastdfs/test 目录下的测试代码，**需要使用make进行编译。**

比如

```
cd ~/tuchuang/fastdfs/test
make
```

```
root@iZbp1h2l856zgoegc8rvnhZ:~/tuchuang/fastdfs/test# make
cc -g -Wall -O -D_FILE_OFFSET_BITS=64 -DDEBUG -c -o common_func.o common_func.c  -I/usr/local/include
cc -g -Wall -O -D_FILE_OFFSET_BITS=64 -DDEBUG -c -o dfs_func.o dfs_func.c  -I/usr/local/include
cc -g -Wall -O -D_FILE_OFFSET_BITS=64 -DDEBUG -o gen_files gen_files.c  common_func.o dfs_func.o -L/usr/l
cal/lib -lfdfsclient -lfastcommon -I/usr/local/include
cc -g -Wall -O -D_FILE_OFFSET_BITS=64 -DDEBUG -o test_upload test_upload.c  common_func.o dfs_func.o -L/u
r/local/lib -lfdfsclient -lfastcommon -I/usr/local/include
cc -g -Wall -O -D_FILE_OFFSET_BITS=64 -DDEBUG -o test_download test_download.c  common_func.o dfs_func.o
L/usr/local/lib -lfdfsclient -lfastcommon -I/usr/local/include
cc -g -Wall -O -D_FILE_OFFSET_BITS=64 -DDEBUG -o test_delete test_delete.c  common_func.o dfs_func.o -L/u
r/local/lib -lfdfsclient -lfastcommon -I/usr/local/include
cc -g -Wall -O -D_FILE_OFFSET_BITS=64 -DDEBUG -o combine_result combine_result.c  common_func.o dfs_func.
 -L/usr/local/lib -lfdfsclient -lfastcommon -I/usr/local/include
root@iZbp1h2l856zgoegc8rvnhZ:~/tuchuang/fastdfs/test# ls
combine_result    common_func.o  dfs_func_pc.c  test_delete     test_download.c   test_upload.c
combine_result.c  dfs_func.c     gen_files      test_delete.c   test_download.sh  test_upload.sh
common_func.c     dfs_func.h     gen_files.c    test_delete.sh  test_types.h
common_func.h     dfs_func.o     Makefile       test_download   test_upload
root@iZbp1h2l856zgoegc8rvnhZ:~/tuchuang/fastdfs/test#
```



3.1 上传测试

对应的测试程序test_upload.c + test_upload.sh

可以模拟十个并发客户端。.0、.1、....、.9是客户端序号而已。

测试结果在fastdfs/test/upload目录。

使用6种不同大小的文件进行测试，分别为

文件规格	测试次数	
5K	100**0000**	
50K	2000000	
200K	1000000	
1M	200000	
10M	20000	
100M	1000	

或者

文件规格	测试次数	
5K	**50000**	
50K	10000	
200K	5000	
1M	500	
10M	50	
100M	10	

（1）单客户端测试本地上传到本机服务器

但这里配置的是外网IP地址

root@iZbp1d83xkvoja33dm7ki2Z:~/0voice/cloud-drive/fastdfs/test# ./test_upload 1

proccess 1, time used: **577s**

```
tracker_server = 114.215.169.66:22122
```

文件规格	上传次数	成功次数	总耗时 ms	平均耗时ms	QPS
5K	5000	5000	79779	15.9	63
50K	1000	1000	59941	59.9	17
200K	500	500	131765	263.5	3.795
1M	50	50	73525	1,470.5	0.680
10M	5	5	76394	15,278.8	0.065
100M	1	1	152530	152,530	

10MB = 10*8Mb / 15.278秒 = 5.3Mbps

（2）120.27.131.197往服务器114.215.169.66 发送

这里的测试结果和（1）一致，**主要受限于网络带宽。**

```
proccess 1, time used: 564s

5K 5000 5000 8107250K 1000 1000 57925
200K 500 500 125415
1M 50 50 71029
10M 5 5 75547
100M 1 1 150387
```

5M的带宽

（3）上传到本地把ip地址修改为内网的再次测试

./test_upload 1
proccess 1, time used: 5s

#file_type total_count success_count time_used(ms)
5K 5000 5000 243
50K 1000 1000 39
200K 500 500 503
1M 50 50 639
10M 5 5 460
100M 1 1 921

文件规格	上传次数	成功次数	总耗时ms	平均耗时ms	QPS(当前测试)	QPS(5M带宽)
5K	5000	5000	243	0.0486	**20576**	63
50K	1000	1000	39	0.039	25,641	17
200K	500	500	503	1.006	994	3.795
1M	50	50	639	12.78	78	0.680
10M	5	5	460	92	10.8	0.065
100M	1	1	921	921	1.08	

PS：和硬盘写入能力有关系。

（4）如果使用集群的方式进行测试?

性能会怎么样？ 本质而言要考虑的点：

- 网络带宽
- 磁盘读写速度
- 文件大小
- 同组storage的个数，因为storage直接要相互同步影响源文件写入的性能。

主要服务器部署好集群。

3.2 下载测试

test_download.c + test_upload.sh，他们依赖test_upload后产生的upload里面的file_id.x文件，里面记录了要下载文件的file id.

./test_download 1

测试结果在fastdfs/test/download目录。

5120（文件大小）
group1/M00/00/44/ctepQmCyRgaIS7LqAAAUAG74ecsAAAADQP7agAAABUA7824285（file_id）

由本地服务器下载

#file_type total_count success_count time_used(ms)
5K 1162088 1162088 574
50K 220357 220357 117
200K 119790 119790 116
1M 12396 12396 8630

3.3 删除测试

test_delete.c + test_delete.sh，他们依赖test_upload后产生的upload里面的file_id.x文件，里面记录了要删除的文件的file id.

./test_delete 1

5120（文件大小）
group1/M00/00/44/ctepQmCyRgaIS7LqAAAUAG74ecsAAAADQP7agAAABUA7824285（file_id）

从本地服务器删除
#file_type total_count success_count time_used(ms)5K 5000 5000 73
50K 1000 1000 9
200K 500 500 6
1M 50 50 1
10M 5 5 5
100M 1 1 15

3.4 测试性能总结

3.4.1 上传文件

提升上传性能的方法：

- 增加group
- 增加带宽
- 使用读写性能高的磁盘

单纯增加每个group的storage只能应对上传峰值，不能从根本上提升上传能力。

3.4.2 下载文件

- 增加storage
- 增加group
- 增加带宽
- 使用读写性能高的磁盘

具体见课上分析。

fastdfs打满千M带宽是很容易的。

4 上云测试注意事项

上云和本地没有太多区别，主要就是注意开发对外的端口。

```
root@iZbp1d83xkvoja33dm7ki2Z:~/0voice/cloud-drive/fastdfs/test# ifconfig eth0:
flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        内外  inet 172.19.47.241  netmask 255.255.192.0  broadcast 172.19.63.255
        inet6 fe80::216:3eff:fe17:9305  prefixlen 64  scopeid 0x20<link>
        ether 00:16:3e:17:93:05  txqueuelen 1000  (Ethernet)
        RX packets 9365809  bytes 3378002224 (3.3 GB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 7581625  bytes 5072239197 (5.0 GB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

120.27.131.197 外网

部署在外网主要是端口开放：80、22122、23000等，可以改成只需要自己家里的**外网ip和公司的外网ip**。

如果云服务器所有的机器都在同一个集群内部，可以不对外开放22122、23000。

5 fastdfs优化策略

见课上分析

最大并发连接数设置

storage.conf
tracker.conf

参数名：max_connections
缺省值：1024
说明：FastDFS采用预先分配好buffer队列的做法，分配的内存大小为： max_connections * buff_size，因此配置的连接数越大，消耗的内存越多。不建议配置得过大，以避免无谓的内存开销。

工作线程数设置

参数名： work_threads （网络IO的工作线程数量）
缺省值：4
说明：为了避免CPU上下文切换的开销，以及不必要的资源消耗，不建议将本参数设置得过大。为了发挥出多个CPU的效能，系统中的线程数总和，应等于CPU总数。

- 对于tracker server，公式为： work_threads + 1 = CPU数
- 对于storage server，公式为： work_threads + 1 + (disk_reader_threads + disk_writer_threads) * store_path_count = CPU数
-

CPU数是4 = work_threads(1) + 1 + disk_reader_threads(1) + disk_writer_threads(4) = 4cpu

同步文件线程
storage–> tracker的线程

storage目录数设置

参数名：subdir_count_per_path

缺省值：256

说明：FastDFS采用**二级目录的做法**，目录会在 FastDFS初始化时自动创建。存储海量小文件， 打开了trunk存储方式的情况下，建议将本参数适当改小，**比如设置为32**，此时存放文件的目录数为 32 * 32 = 1024。假如trunk文件大小采用缺省值**64MB**，磁盘空间为2TB，那么每个目录下存放的trunk文件数均值为：**2TB / (1024 * 64MB) = 32**个。

4TB / (1024 * 64MB) = 64个。

8T / **(1024 * 64MB) = 128** 个 –》8T / **(64*64* 64MB) = 32个**

二级目录：

256 + 256 + 150 =662

一级目录

256 + 3,906 = 4,162

storage磁盘读写线程设置

- disk_rw_separated：磁盘读写是否分离
- disk_reader_threads：单个磁盘读线程数
- disk_writer_threads：单个磁盘写线程数
- 如果磁盘读写混合，单个磁盘读写线程数为读线程数和写线程数之和
- **对于单盘挂载方式，磁盘读写线程分别设置为1即可**
- 如果磁盘做了RAID，那么需要酌情加大读写线程数，这样才能最大程度地发挥磁盘性能

storage同步延迟相关设置

- sync_binlog_buff_interval：将binlog buffer写入磁盘的时间间隔，取值大于0，缺省值为60s
- sync_wait_msec：如果没有需要同步的文件，对binlog进行轮询的时间间隔，取值大于0，缺省值为100ms
- sync_interval：同步完一个文件后，休眠的毫秒数，缺省值为0

为了缩短文件同步时间，可以将上述3个参数适当调小即可

6 磁盘性能监测

iostat

```
// kb/s显示磁盘信息，每2s刷新一次

iostat -d -k 2

// kb/s显示磁盘统计信息及扩展信息，每1s刷新 ，刷新10次结束

iostat -dkx 1 10
```

磁盘的统计参数

- **tps**：该设备每秒的传输次数（Indicate the number of transfers per second that were issued to the device.）。"一次传输"意思是"一次I/O请求"。
- 多个逻辑请求可能会被合并为"一次I/O请求"。"一次传输"请求的大小是未知的。
- kB_read/s：每秒从设备（drive expressed）读取的数据量；
- kB_wrtn/s：每秒向设备（drive expressed）写入的数据量；
- kB_read：读取的总数据量；
- kB_wrtn：写入的总数量数据量；这些单位都为Kilobytes。
-
- rrqm/s: 每秒对该设备的读请求被合并次数，文件系统会对读取同块(block)的请求进行合并
- wrqm/s: 每秒对该设备的写请求被合并次数
- r/s: 每秒完成的读次数
- w/s: 每秒完成的写次数
- rkB/s: 每秒读数据量(kB为单位)
- wkB/s: 每秒写数据量(kB为单位)
- avgrq-sz:平均每次IO操作的数据量(扇区数为单位)
- avgqu-sz: 平均等待处理的IO请求队列长度
- await: 平均每次IO请求等待时间(包括等待时间和处理时间，毫秒为单位)
- svctm: 平均每次IO请求的处理时间(毫秒为单位)
- %util: 采用周期内用于IO操作的时间比率，即IO队列非空的时间比率

7 参考阅读

- 海量小文件问题综述 https://blog.csdn.net/liuaigui/article/details/9981135
- FastDFS防盗链 https://www.cnblogs.com/xiaolinstudy/p/9341779.html
- FASTDFS文件服务器架构方案分析 https://wenku.baidu.com/view/3d0987452dc58bd63186bceb19e8b8f67c1cefb0.html
- 基于FastDFS的云存储文件系统性能优化设计与实现 http://www.doc88.com/p-5843806382095.html
- 基于FastDFS的目录文件系统的+研究与实现 https://www.doc88.com/p-5327654549305.html?s=rel&id=2
- 基于FastDFS云存储系统的研究与设计 https://www.doc88.com/p-7098966281892.html?s=rel&id=1
- 基于FastDFS架构的小文件存储系统的设计与实现 https://www.doc88.com/p-7098966281892.html?s=rel&id=1 （课题来源于腾讯实时生培训项目）
- 基于FastDFS的大并发问题的研究与应用 https://www.doc88.com/p-9813549702927.html?s=like&id=1

已知使用FastDFS的用户• 某大型网盘（公司名对方要求保密）

- UC（http://www.uc.cn/）
- 支付宝（http://www.alipay.com/）
- 京东商城（http://www.360buy.com/）
- 淘淘搜（http://www.taotaosou.com/）
- 飞信（http://feixin.10086.cn/）
- 赶集网（http://www.ganji.com/）
- 淘米网（http://www.61.com/）
- 迅雷（http://www.xunlei.com/）
- 蚂蜂窝（http://www.mafengwo.cn/）
- 丫丫网（http://www.iyaya.com/）
- 虹网（http://3g.ahong.com/）
- 5173（http://www.5173.com/）
- 华夏原创网（http://www.yuanchuang.com/）
- 华师京城教育云平台（http://www.hsjdy.com.cn/）
- 视友网（http://www.cuctv.com/）
- 。。。