fastdfs 原理与过程

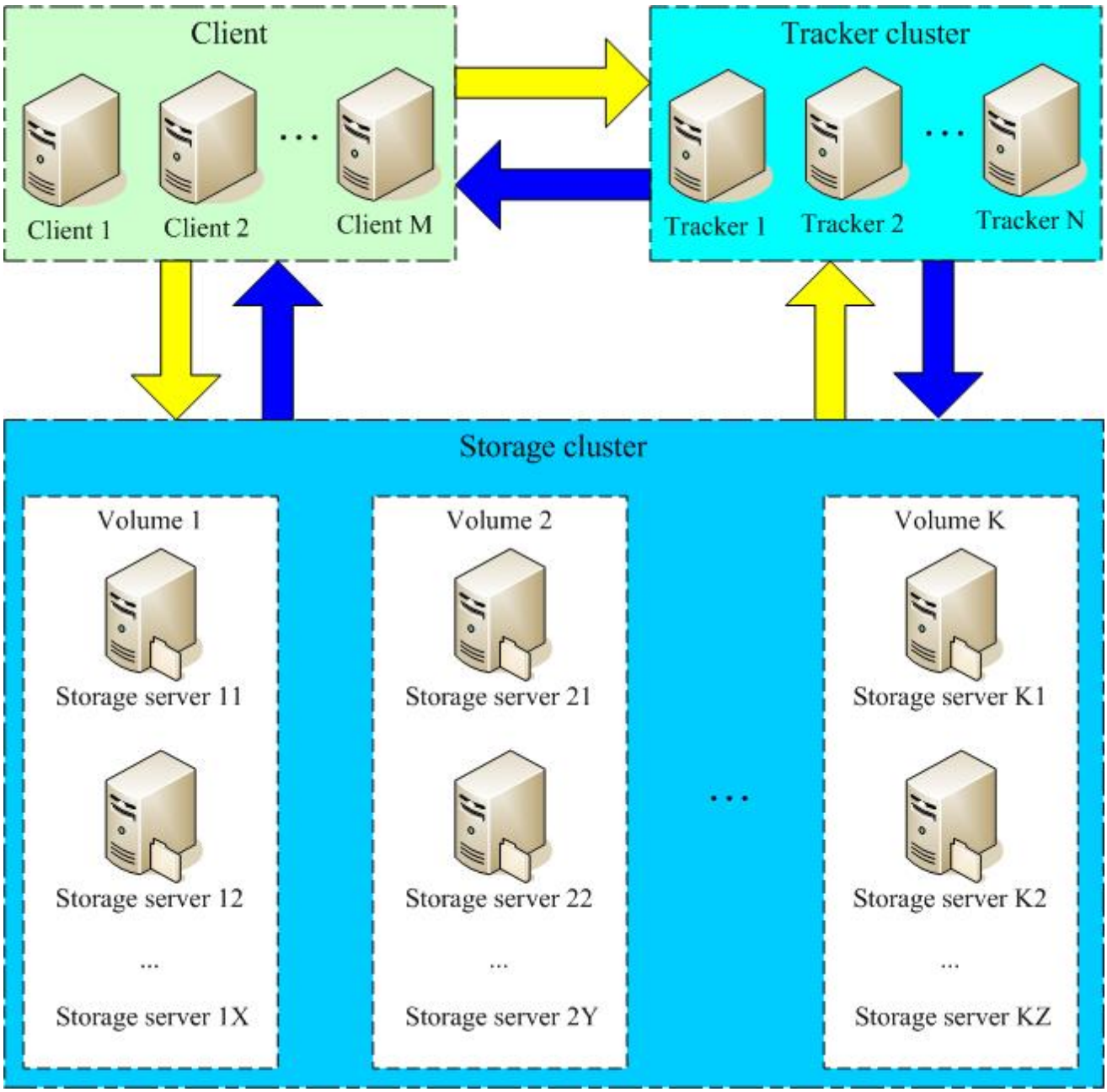
前言：

（1）每次上传文件后都会返回一个地址，用户需要自己保存此地址。

（2）为了支持大容量，存储节点（服务器）采用了分卷（或分组）的组织方式。存储系统由一个或多个卷组成，卷与卷之间的文件是相互独立的，所有卷的文件容量累加就是整个存储系统中的文件容量。一个卷可以由一台或多台存储服务器组成，一个卷下的存储服务器中的文件都是相同的，卷中的多台存储服务器起到了冗余备份和负载均衡的作用。

网摘1

FastDFS是一个开源的轻量级分布式文件系统，由跟踪服务器（tracker server）、存储服务器（storage server）和客户端（client）三个部分组成，主要解决了海量数据存储问题，特别适合以中小文件（建议范围：4KB < file_size <500MB）为载体的在线服务。



Storage server

Storage server（后简称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，所以建议group内的多个storage尽量配置相同，以免造成存储空间的浪费。

以group为单位组织存储能方便的进行应用隔离、负载均衡、副本数定制（group内storage server数量即为该group的副本数），比如将不同应用数据存到不同的group就能隔离应用数据，同时还可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内地其他机器，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可配置多个数据存储目录，比如有10块磁盘，分别挂载在/data/disk1-/data/disk10，则可将这10个目录都配置为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（后面会介绍），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数太多，在storage第一次启动时，会在每个数据存储目录里创建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。

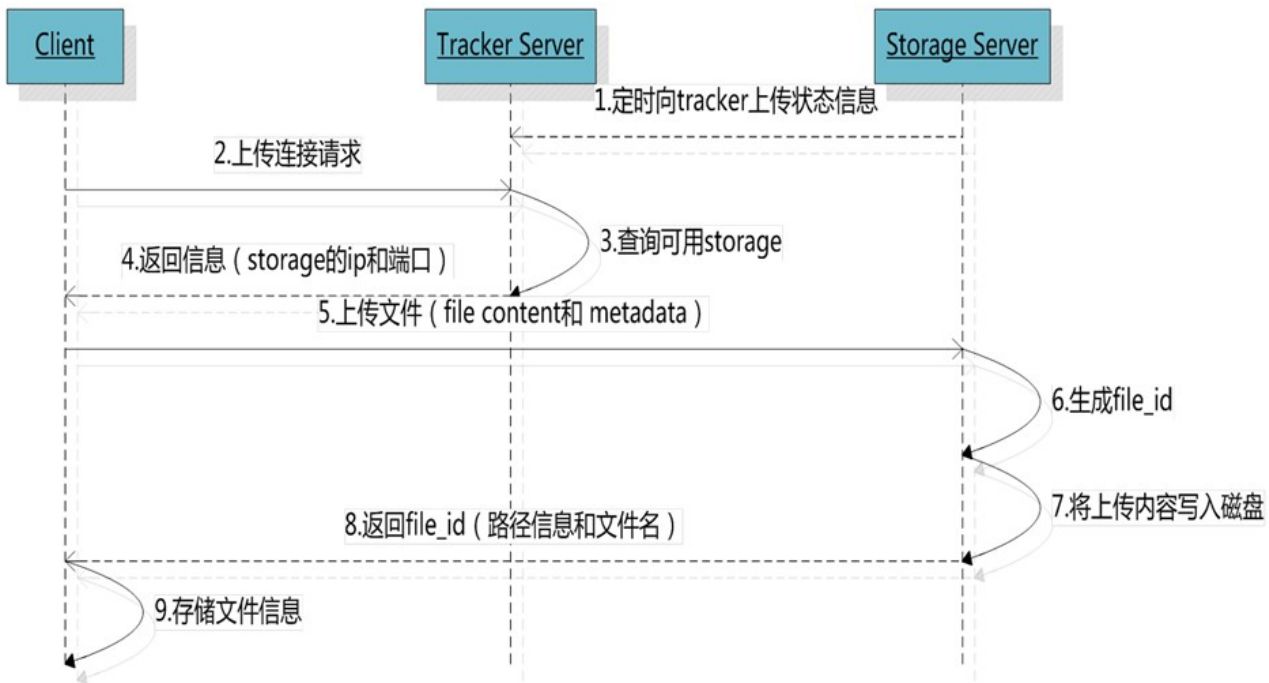
Tracker server

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，并保持周期性的心跳，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；另外tracker上的元信息都是由storage汇报的信息生成的，本身不需要持久化任何数据，这样使得tracker非常容易扩展，直接增加tracker机器即可扩展为tracker cluster来服务，cluster里每个tracker之间是完全对等的，所有的tracker都接受stroage的心跳信息，生成元数据信息来提供读写服务。

Upload file

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



- 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trakcer。

- 选择存储的group

当tracker接收到upload file的请求时，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： 1. Round robin，所有的group间轮询 2. Specified group，指定某一个确定的group 3. Load balance，剩余存储空间多多group优先

- 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：

1. Round robin，在group内的所有storage间轮询
2. First server ordered by ip，按ip排序
3. First server ordered by priority，按优先级排序（优先级在storage上配置）

- 选择storage path

当分配好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：

1. Round robin，多个存储目录间轮询
2. 剩余存储空间最多的优先

- 生成Fileid

选定存储目录之后，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。

- 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（猜测），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。

- 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，主要用于区分文件类型）拼接而成。

文件同步

写文件时，客户端将文件写至group内一个storage server即认为写文件成功，storage server写完文件后，会由后台线程将文件同步至同group内其他的storage server。

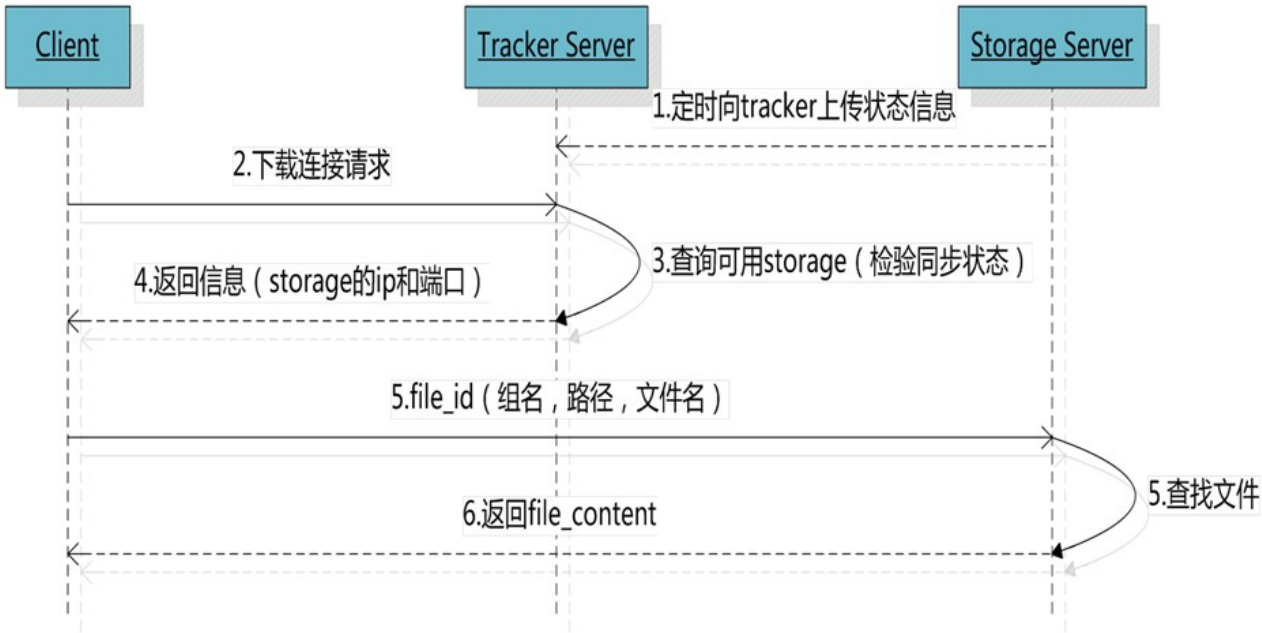
每个storage写文件后，同时会写一份binlog，binlog里不包含文件数据，只包含文件名等元信息，这份binlog用于后台同步，storage会记录向group内其他storage同步的进度，以便重启后能接上次的进度继续同步；进度以时间戳的方式进行记录，所以最好能保证集群内所有server的时钟保持同步。

storage的同步进度会作为元数据的一部分汇报到tracker上，tracke在选择读storage的时候会以同步进度作为参考。

比如一个group内有A、B、C三个storage server，A向C同步到进度为T1 (T1以前写的文件都已经同步到B上了），B向C同步到时间戳为T2（T2 > T1)，tracker接收到这些同步进度信息时，就会进行整理，将最小的那个做为C的同步时间戳，本例中T1即为C的同步时间戳为T1（即所有T1以前写的数据都已经同步到C上了）；同理，根据上述规则，tracker会为A、B生成一个同步时间戳。

Download file

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。

tracker发送download请求给某个tracker，必须带上文件名信息，tracke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，所以有可能出现在读到时候，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) > 同步延迟阀值（如一天）。 - 经过同步延迟阈值时间，认为文件肯定已经同步了。

小文件合并存储

将小文件合并存储主要解决如下几个问题：

1. 本地文件系统inode数量有限，从而存储的小文件数量也就受到限制。
2. 多级目录+目录里很多文件，导致访问文件的开销很大（可能导致很多次IO）
3. 按小文件存储，备份与恢复的效率低

FastDFS在V3.0版本里引入小文件合并存储的机制，可将多个小文件存储到一个大的文件（trunk file），为了支持这个机制，FastDFS生成的文件fileid需要额外增加16个字节

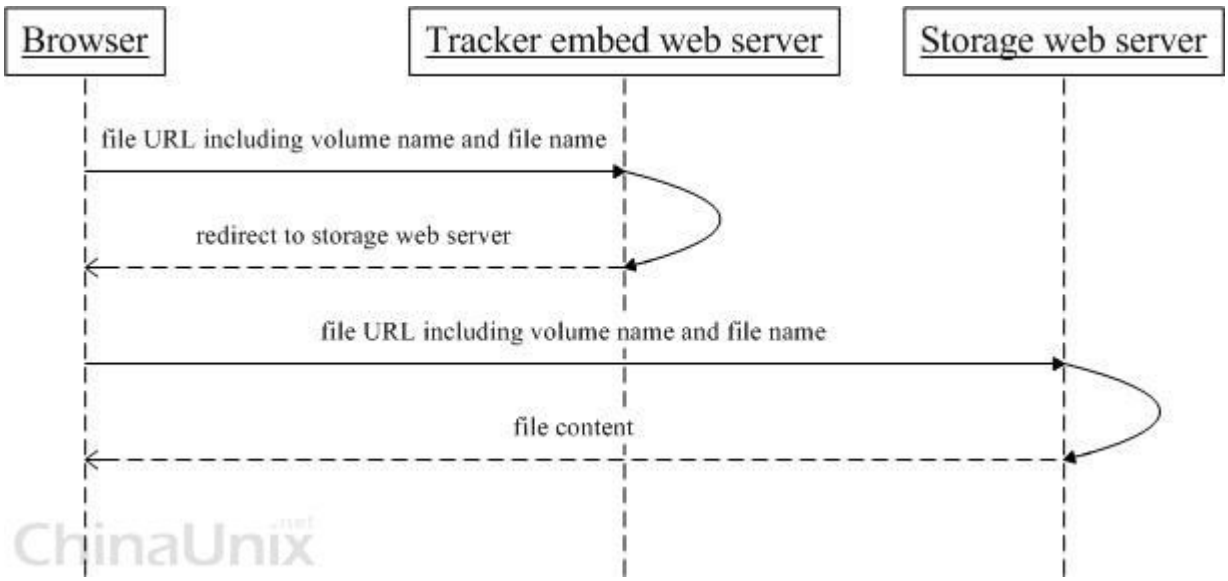
1. trunk file id
2. 文件在trunk file内部的offset
3. 文件占用的存储空间大小 （字节对齐及删除空间复用，文件占用存储空间>=文件大小）

每个trunk file由一个id唯一标识，trunk file由group内的trunk server负责创建（trunk server是tracker选出来的），并同步到group内其他的storage，文件存储合并存储到trunk file后，根据其offset就能从trunk file读取到文件。

文件在trunk file内的offset编码到文件名，决定了其在trunk file内的位置是不能更改的，也就不能通过compact的方式回收trunk file内删除文件的空间。但当trunk file内有文件删除时，其删除的空间是可以被复用的，比如一个100KB的文件被删除，接下来存储一个99KB的文件就可以直接复用这片删除的存储空间。

HTTP访问支持

FastDFS的tracker和storage都内置了http协议的支持，客户端可以通过http协议来下载文件，tracker在接收到请求时，通过http的redirect机制将请求重定向至文件所在的storage上；除了内置的http协议外，FastDFS还提供了通过apache或nginx扩展模块下载文件的支持。



其他特性

FastDFS提供了设置/获取文件扩展属性的接口（setmeta/getmeta），扩展属性以key-value对的方式存储在storage上的同名文件（拥有特殊的前缀或后缀），比如/group/M00/00/01/some_file为原始文件，则该文件的扩展属性存储在/group/M00/00/01/.some_file.meta文件（真实情况不一定是这样，但机制类似），这样根据文件名就能定位到存储扩展属性的文件。

以上两个接口作者不建议使用，额外的meta文件会进一步"放大"海量小文件存储问题，同时由于meta非常小，其存储空间利用率也不高，比如100bytes的meta文件也需要占用4K（block_size）的存储空间。

FastDFS还提供appender file的支持，通过upload_appender_file接口存储，appender file允许在创建后，对该文件进行append操作。实际上，appender file与普通文件的存储方式是相同的，不同的是，appender file不能被合并存储到trunk file。

问题讨论

从FastDFS的整个设计看，基本上都已简单为原则。比如以机器为单位备份数据，简化了tracker的管理工作；storage直接借助本地文件系统原样存储文件，简化了storage的管理工作；文件写单份到storage即为成功、然后后台同步，简化了写文件流程。但简单的方案能解决的问题通常也有限，FastDFS目前尚存在如下问题（欢迎探讨）。

- 数据安全性
- 写一份即成功：从源storage写完文件至同步到组内其他storage的时间窗口内，一旦源storage出现故障，就可能导致用户数据丢失，而数据的丢失对存储系统来说通常是不可接受的。

- 缺乏自动化恢复机制：当storage的某块磁盘故障时，只能换存磁盘，然后手动恢复数据；由于按机器备份，似乎也不可能有自动化恢复机制，除非有预先准备好的热备磁盘，缺乏自动化恢复机制会增加系统运维工作。
- 数据恢复效率低：恢复数据时，只能从group内其他的storage读取，同时由于小文件的访问效率本身较低，按文件恢复的效率也会很低，低的恢复效率也就意味着数据处于不安全状态的时间更长。
- 缺乏多机房容灾支持：目前要做多机房容灾，只能额外做工具来将数据同步到备份的集群，无自动化机制。
- 存储空间利用率
- 单机存储的文件数受限于inode数量
- 每个文件对应一个storage本地文件系统的文件，平均每个文件会存在block_size/2的存储空间浪费。
- 文件合并存储能有效解决上述两个问题，但由于合并存储没有空间回收机制，删除文件的空间不保证一定能复用，也存在空间浪费的问题
- 负载均衡
- group机制本身可用来做负载均衡，但这只是一种静态的负载均衡机制，需要预先知道应用的访问特性；同时group机制也导致不可能在group之间迁移数据来做动态负载均衡。

网摘2

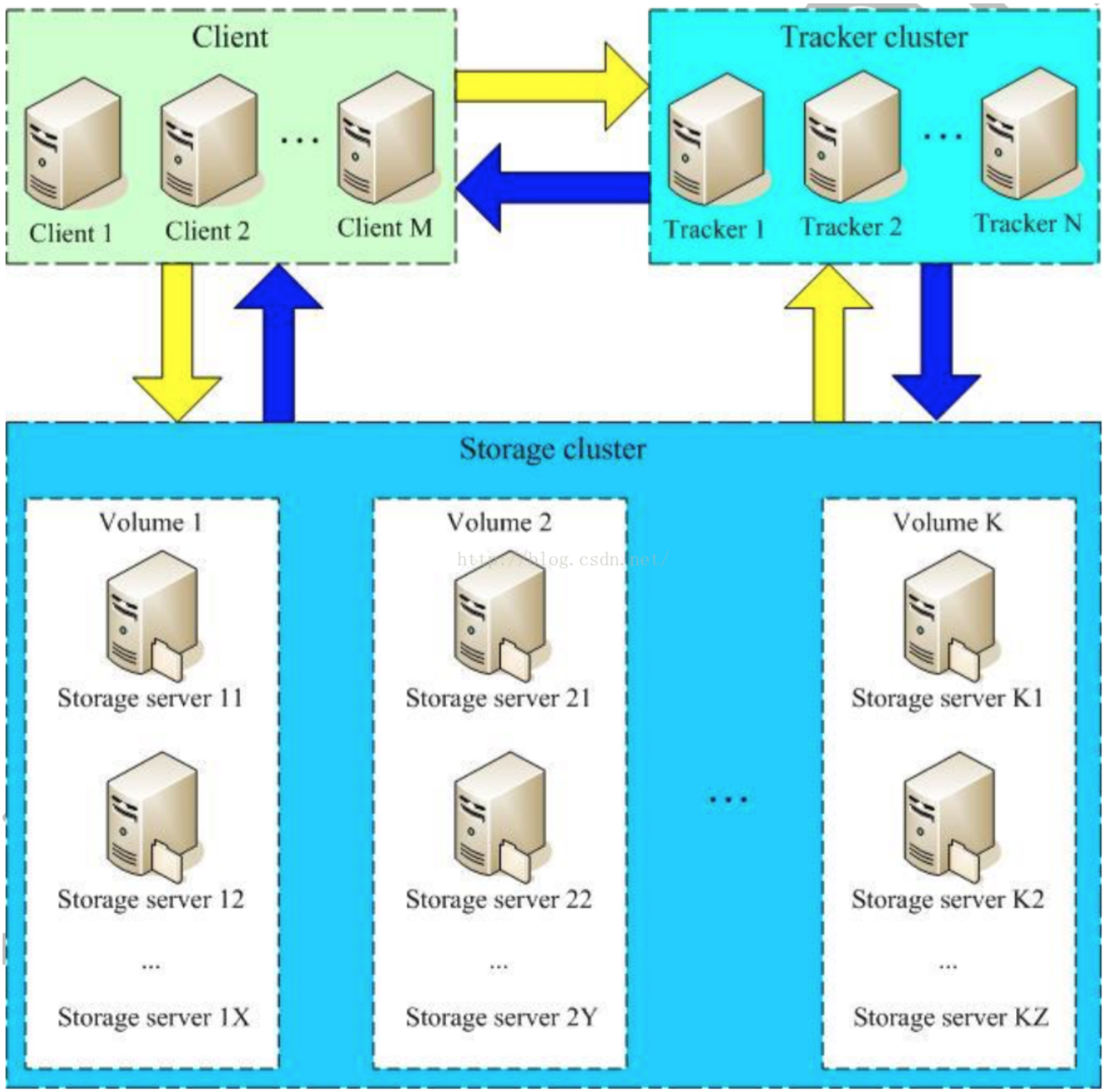
FastDFS 介绍

FastDFS 是一个 C 语言实现的开源轻量级分布式文件系统,作者余庆(happyfish100),支持 Linux、FreeBSD、AID 等 Unix 系统,解决了大数据存储和读写负载均衡等问题,适合存储 4KB~500MB 之间的小文件,如图片网站、短视频网站、文档、app 下载站等,UC、京东、支付宝、迅雷、酷狗等都有使用,其中 UC 基于 FastDFS 向用户提供网盘、广告和应用下载的业务的存储服务 FastDFS 与 MogileFS、HDFS、TFS 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。

FastDFS 架构

FastDFS服务有三个角色:跟踪服务器(tracker server)、存储服务器(storage server)和客户端(client)
tracker server: 跟踪服务器,主要做调度工作,起到均衡的作用;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 group 等信息,并保持周期性心跳, Tracker根据storage心跳信息,建立group--->[storage server list]的映射表;tracker管理的元数据很少,会直接存放在内存;tracker 上的元信息都是由 storage 汇报的信息生成的,本身不需要持久化任何数据,tracker 之间是对等关系,因此扩展 tracker 服务非常容易,之间增加 tracker服务器即可,所有tracker都接受stroage心跳信息,生成元数据信息来提供读写服务(与 其他 Master-Slave 架构的优势是没有单点,tracker 也不会成为瓶颈,最终数据是和一个可用的 Storage Server进行传输的)
storage server:存储服务器,主要提供容量和备份服务;以 group 为单位,每个 group 内可以包含多台storage server,数据互为备份,存储容量空间以group内容量最小的storage为准;建 议group内的storage server配置相同;以group为单位组织存储能够方便的进行应用隔离、负载均衡和副本数定制;缺点是 group 的容量受单机存储容量的限制,同时 group 内机器坏掉,数据 恢复只能依赖 group 内其他机器重新同步(坏盘替换,重新挂载重启 fdfs_storaged 即可)

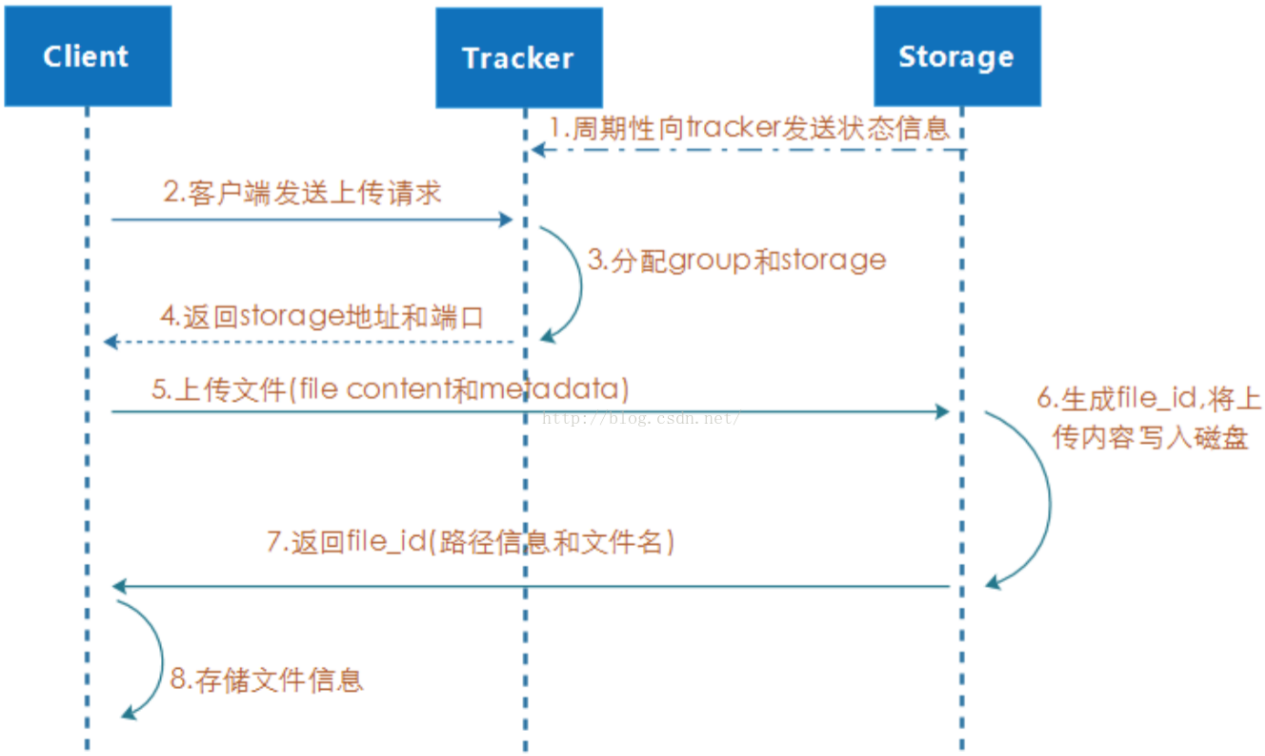
多个group之间的存储方式有3种策略:round robin(轮询)、load balance(选择最大剩余空 间的组上传文件)、specify group(指定group上传)
group 中 storage 存储依赖本地文件系统,storage 可配置多个数据存储目录,磁盘不做 raid, 直接分别挂载到多个目录,将这些目录配置为 storage 的数据目录即可
storage 接受写请求时,会根据配置好的规则,选择其中一个存储目录来存储文件;为避免单 个目录下的文件过多,storage 第一次启时,会在每个数据存储目录里创建 2 级子目录,每级 256 个,总共 65536 个,新写的文件会以 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



总结:1.高可靠性:无单点故障 2.高吞吐性:只要Group足够多,数据流量是足够分散的

FastDFS 工作流程

上传



FastDFS上传文件流程

FastDFS 提供基本的文件访问接口,如 upload、download、append、delete 等

选择tracker server

集群中 tracker 之间是对等关系,客户端在上传文件时可用任意选择一个 tracker

选择存储 group

当tracker接收到upload file的请求时,会为该文件分配一个可以存储文件的group,目前支持选择 group 的规则为:

1. Round robin,所有 group 轮询使用
2. Specified group,指定某个确定的 group
3. Load balance,剩余存储空间较多的 group 优先

选择storage server

当选定group后,tracker会在group内选择一个storage server给客户端,目前支持选择server 的规则为:

1. Round robin,所有 server 轮询使用(默认)
2. 根据IP地址进行排序选择第一个服务器(IP地址最小者)
3. 根据优先级进行排序(上传优先级由storage server来设置,参数为upload_priority)

选择storage path(磁盘或者挂载点)

当分配好storage server后,客户端将向storage发送写文件请求,storage会将文件分配一个数据存储目录,目前支持选择存储路径的规则为:

1. round robin,轮询(默认)
2. load balance,选择使用剩余空间最大的存储路径

选择下载服务器

目前支持的规则为：

1. 轮询方式,可以下载当前文件的任一storage server
2. 从源storage server下载

生成 file_id

选择存储目录后,storage 会生成一个 file_id,采用 Base64 编码,包含字段包括:storage server ip、文件创建时间、文件大小、文件 CRC32 校验码和随机数;每个存储目录下有两个 256*256 个子目录,storage 会按文件 file_id 进行两次 hash,路由到其中一个子目录,,然后将文件以 file_id 为文件名存储到该子目录下,最后生成文件路径:group 名称、虚拟磁盘路径、数据两级目录、file_id

其中,

组名:文件上传后所在的存储组的名称,在文件上传成功后由存储服务器返回,需要客户端自行保存

虚拟磁盘路径:存储服务器配置的虚拟路径,与磁盘选项 store_path*参数对应

数据两级目录:存储服务器在每个虚拟磁盘路径下创建的两级目录,用于存储数据文件

同步机制

1. 新增tracker服务器数据同步问题

由于 storage server 上配置了所有的 tracker server. storage server 和 trackerserver 之间的通信是由 storage server 主动发起的,storage server 为每个 tracker server 启动一个线程进行通信;在通信过程中,若发现该 tracker server 返回的本组 storage server列表比本机记录少,就会将该tracker server上没有的storage server 同步给该 tracker,这样的机制使得 tracker 之间是对等关系,数据保持一致

2. 新增storage服务器数据同步问题

若新增storage server或者其状态发生变化,tracker server都会将storage server列表同步给该组内所有 storage server;以新增 storage server 为例,因为新加入的 storage server 会主动连接 tracker server,tracker server 发现有新的 storage server 加入,就会将该组内所有的 storage server 返回给新加入的 storage server,并重新将该组的 storage server列表返回给该组内的其他storage server;

3. 组内storage数据同步问题

组内storage server之间是对等的,文件上传、删除等操作可以在组内任意一台storageserver 上进行。文件同步只能在同组内的 storage server 之间进行,采用 push 方式, 即源服务器同步到目标服务器

A. 只在同组内的storage server之间进行同步
B. 源数据才需要同步,备份数据不再同步
C. 特例:新增storage server时,由其中一台将已有所有数据(包括源数据和备份数据)同步到新增服务器

storage server的7种状态:

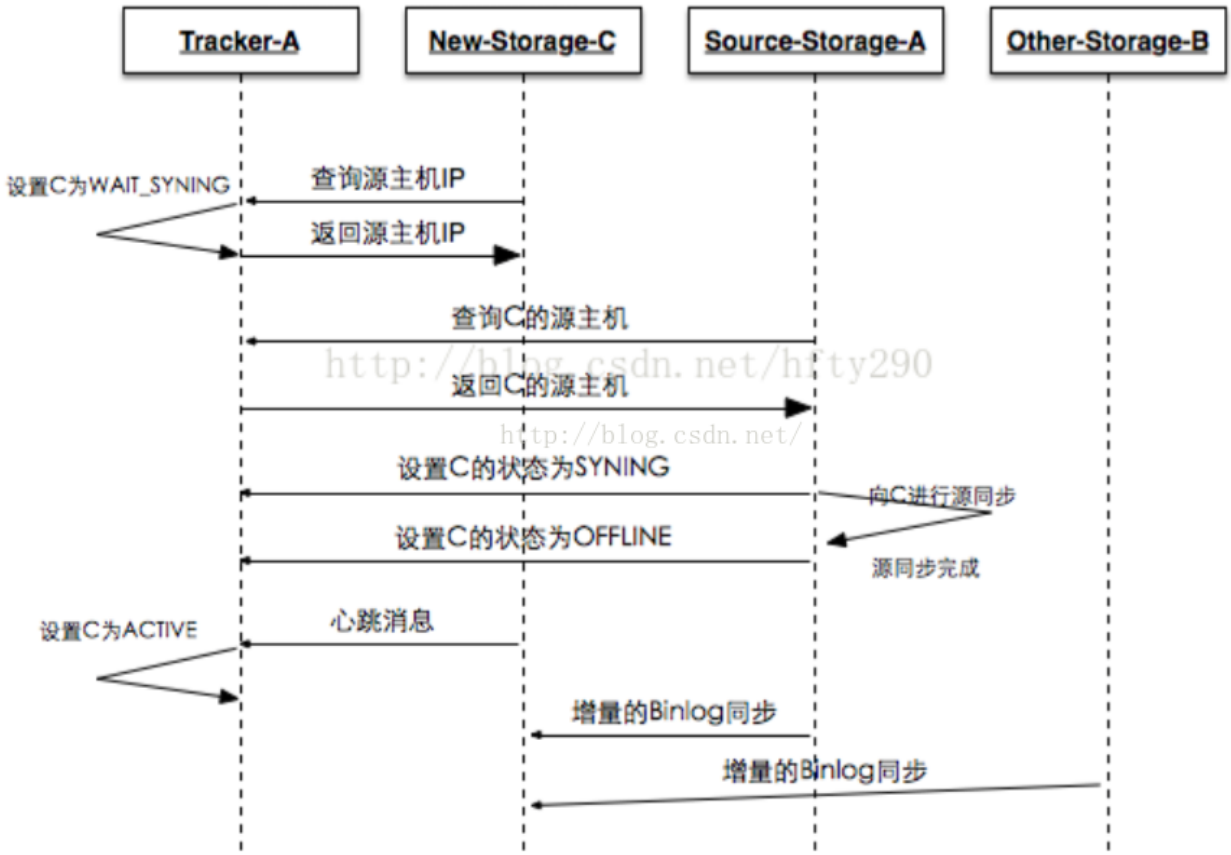
通过命令 fdfs_monitor /etc/fdfs/client.conf 可以查看 ip_addr 选项显示 storage server 当前状态

- INIT : 初始化,尚未得到同步已有数据的源服务器
- WAIT_SYNC : 等待同步,已得到同步已有数据的源服务器
- SYNCING : 同步中
- DELETED : 已删除,该服务器从本组中摘除
- OFFLINE :离线

- ONLINE : 在线,尚不能提供服务
- ACTIVE : 在线,可以提供服务

组内增加storage serverA状态变化过程:

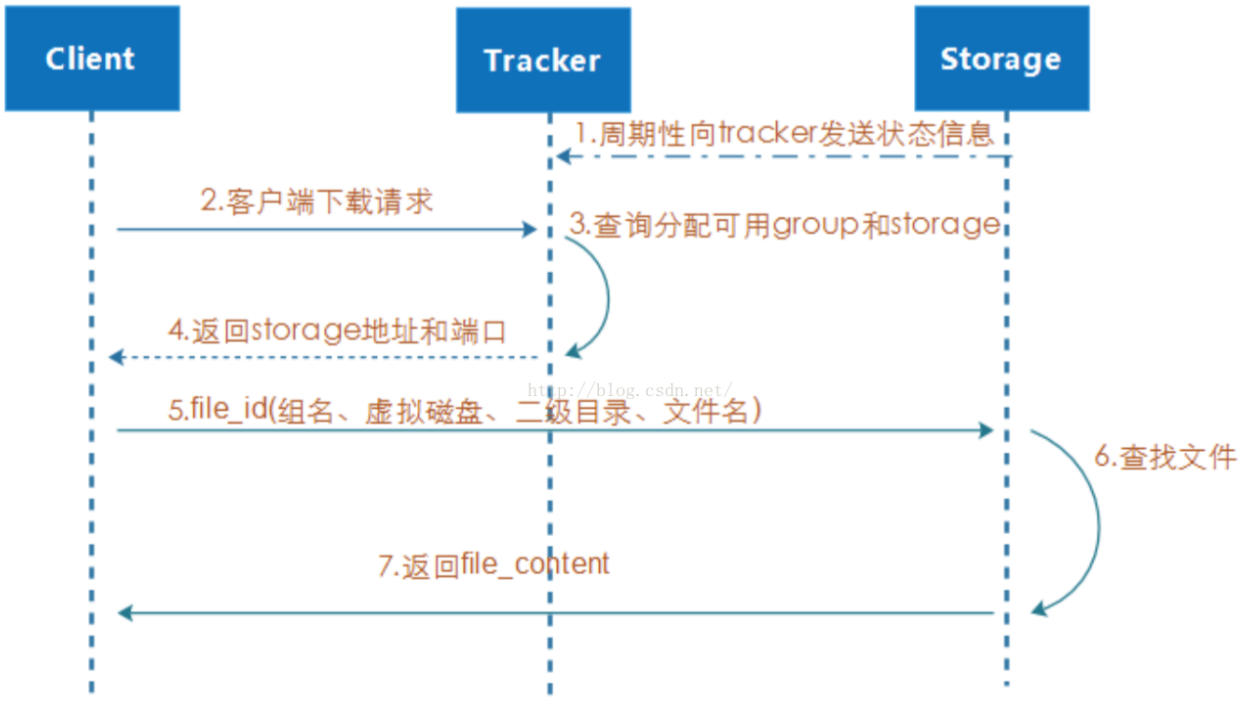
1. storage server A 主动连接 tracker server,此时 tracker server 将 storageserverA 状态设置为 INIT
2. storage server A 向 tracker server 询问追加同步的源服务器和追加同步截止时间点(当前时间),若组内只有 storage server A或者上传文件数为0,则告诉新机器不需要数据同步,storage server A 状态设置为 ONLINE ;若组内没有 active 状态机器,就返回错误给新机器,新机器睡眠尝试;否则 tracker 将其状态设置为 WAIT_SYNC
3. 假如分配了 storage server B 为同步源服务器和截至时间点,那么 storage server B会将截至时间点之前的所有数据同步给storage server A,并请求tracker设置 storage server A 状态为SYNCING;到了截至时间点后,storage server B向storage server A 的同步将由追加同步切换为正常 binlog 增量同步,当取不到更多的 binlog 时,请求tracker将storage server A设置为OFFLINE状态,此时源同步完成
4. storage server B 向 storage server A 同步完所有数据,暂时没有数据要同步时, storage server B请求tracker server将storage server A的状态设置为ONLINE
5. 当 storage server A 向 tracker server 发起心跳时,tracker sercer 将其状态更改为 ACTIVE,之后就是增量同步(binlog)



注释:

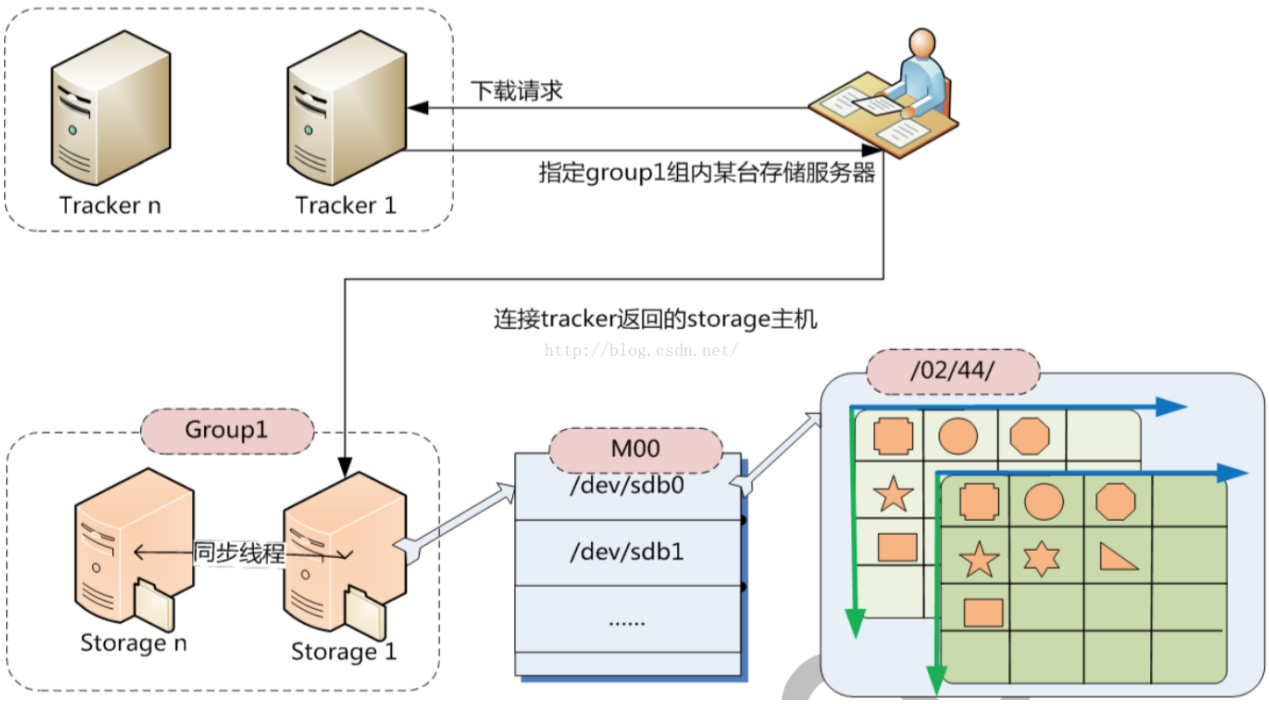
1.整个源同步过程是源机器启动一个同步线程,将数据 push 到新机器,最大达到一个磁盘的 IO,不能并发
2.由于源同步截止条件是取不到 binlog,系统繁忙,不断有新数据写入的情况,将会导致一直无法完成源同步过程

下载



FastDFS下载文件流程

client 发送下载请求给某个 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 用于读请求;由于 group 内的文件同步在后台是异步进行的,可能出现文件没有同步到其他storage server上或者延迟的问题, 后面我们在使用 nginx_fastdfs_module 模块可以很好解决这一问题



文件合并原理

小文件合并存储主要解决的问题:

1. 本地文件系统inode数量有限,存储小文件的数量受到限制
2. 多级目录+目录里很多文件,导致访问文件的开销很大(可能导致很多次IO)
3. 按小文件存储,备份和恢复效率低

FastDFS 提供合并存储功能,默认创建的大文件为 64MB,然后在该大文件中存储很多小文件; 大文件中容纳一个小文件的空间称作一个 Slot,规定 Slot 最小值为 256 字节,最大为 16MB,即小于 256 字节的文件也要占用 256 字节,超过 16MB 的文件独立存储;
为了支持文件合并机制,FastDFS生成的文件file_id需要额外增加16个字节;每个trunk file 由一个id唯一标识,trunk file由group内的trunk server负责创建(trunk server是tracker 选出来的),并同步到group内其他的storage,文件存储合并存储到trunk file后,根据其文件偏移量就能从trunk file中读取文件