Google 文件系统-GFS

ZhaohengLi 2017050025

2020 年 4 月 21 日

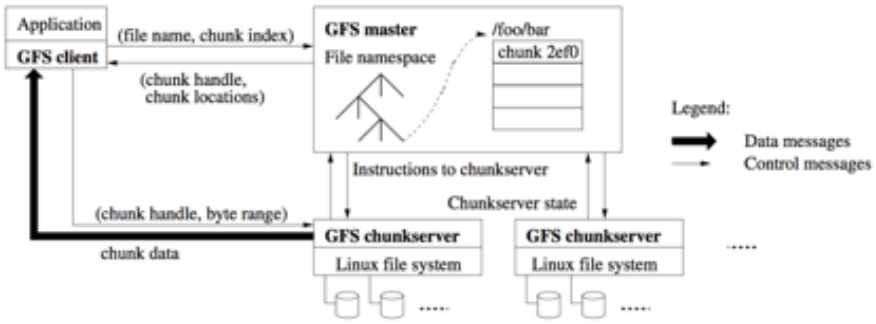
1 背景

Google File System(简称 GFS) 是适用于大规模且可扩展的分布式文件系统，可以部署在廉价的商务服务器上，在保证系统可靠性和可用性的同时，大大降低了系统的成本。GFS 的设计目标是高性能、高可靠、高可用性。

GFS 把机器故障视为正常现象，可以很好地处理系统故障。GFS 系统通常会部署在上百台甚至上千台廉价服务器上，并会有相当多台廉价服务器上部署 GFS Client 来访问 GFS 服务，所以应用故障、操作系统 bug、连接故障、网络故障、甚至机器供电故障都是经常发生的故障。GFS 系统可以支持系统监控、故障检测、故障容忍和自动恢复，提供了非常高的可靠性。其次，GFS 系统中的文件一般都是大文件，且文件操作大部分场景下都是 append 而不是 overwrite。一旦文件写入完成后，大部分操作都是读文件且是顺序读。

GFS 提供了非标准（比如 POSIX）的文件系统接口，支持 create、delete、open、close、read 以及 write。另外 GFS 支持 snapshot 和 record append 操作。snapshot 可以以很低的代价创建文件或者目录树的拷贝，record append 可以支持多个 client 并发地向同一个文件 append data，同时还能保证每个 client 的 append 操作的原子性。

2 架构



GFS 系统包括 master、多个 chunkserver 以及多个 client。文件被切分为固定大小的小文件 (chunk)。每个 chunk 创建时，master 会分配一个 64bit 的全局唯一且不可修改的 chunk handler 来标志这个 chunk。chunkserver 负责将 chunk 存储在本地磁盘的 Linux 文件中，并通过 chunk hander 和 byte range 来读

写 chunk 文件。为了提高可靠性，每个 chunk 都是多副本存储在多个 chunkserver 上，默认情况下是三副本。用户可以为不同名字空间下的文件配置不同的副本数。

master 记录了文件系统的 metadata，包括名字空间、权限控制信息、文件到 chunk 的 mapping 以及 chunk 的分布。master 也负责 chunk 的 lease 管理、无用 chunk 的垃圾回收、chunk 迁移等。master 定期与 chunkserver 通信，向 chunkserver 发送指令并搜集 chunkserver 的状态。GFS client 通过 GFS 的 API 与 GFS 系统通信（读写数据）。client 向 master 请求获取 metadata，真正的读写数据是直接与 chunkserver 交互。client 和 chunkserver 都不 cache 文件数据。因为大部分应用都是基于 API 来 streaming read 大文件且系统的文件数据太多，所以 client 缓存文件数据没有意义。chunkserver 所在机器的 Linux 的 buffer cache 以及 cache 了频繁访问的数据，chunkserver 也是没有去 cache 文件数据的。

3 Single Master

单点 master 大大简化了系统设计，因为 master 知晓所有的 meta 信息，所以可以执行更加复杂的 chunk 位置分配和副本策略。但是，在读写数据时必须降低 master 的参与，以避免单点的 master 称为系统瓶颈。client 不会通过 master 来读写文件数据，但是 client 会向 master 发送查询 chunk 位置分布的请求，然后 client 端缓存 chunk 的分布信息，然后直接向 chunkserver 读写数据。大致的读过程如下：

1、client 根据文件名、byte offset 以及 chunk size 计算出要读取的文件的 chunk index

2、client 通过文件名、chunk index 向 master 查询 chunk 的分布

3、master 回复 chunk handler 以及副本分布

4、client 缓存 chunk 的 meta 信息，key 由文件名和 chunk index 组成

5、client 从 chunk 的分布信息中查找距离自己最新的 chunkserver，并发送查询请求。查询请求中包括 chunk hander 以及 byte range。后续对相同 chunk 的查询不需要再次向 master 查询 meta 信息，因为 client 已经缓存了 meta 信息。

4 Chunk Size

chunk size 是 GFS 系统的关键参数，通常设置为 64MB，远大于文件系统的 block 大小。每个 chunk 的副本都 chunkserver 所在机器上以 Linux file 存储。之所为将 chunk size 定为 64MB，主要有以下考虑：

1、可以减少 client 访问 master 查询 meta 信息的次数，降低 master 的访问压力。因为 chunk size 设计比较大，顺序访问一个超大文件时因为 chunk 数较少且 client 缓存了 chunk meta 信息，所以访问 master 的次数就会降低。甚至，client 可以缓存所有文件的 chunk 的 meta 信息，就算是随机读文件，master 也不会成为系统性能瓶颈。

2、可以减少网络开销，保持 client 与 chunkserver 的 TCP 连接，可以执行更多的 chunk 操作。

3、可以减少 master 上需要在内存中记录的 meta data 数据量，降低 master 的内存占用。

size 大的缺点是：小文件包含很少的 chunk，甚至只有一个。这样的话，在多个 client 高并发查询该小文件时对应的 chunk 会成为热点。实际上，这种情况在 GFS 系统中很少发生，因为大部分 client 的操作都是顺序读大文件。但是，考虑以下场景，我们部署一个服务的二进制文件到 GFS 系统中，然后数百台的服务器同时查询二进制文件并启动服务，此时该二进制文件副本所在的 chunkserver 立马就会成为查询瓶颈。当然，可以通过增加副本数和分散服务器的查询时间来解决这种场景下的问题。

5 Metadata

master 主要存储三种类型的 metadata：file 和 chunk 的名字空间，file 到 chunk 的 mapping 信息以及 chunk 的副本分布。所有的 metadata 都在 master 的内存中存储。前两种 meta 信息可以持久化存储，将操作日志存储在 master 的本地磁盘以及将备份日志存储在远端机器上。master 不持久化存储 chunk 的副本分布信息，而是通过与 chunkserver 交互来获取 chunkserver 上的 chunk 信息。

5.1 in-memory data structure

meta 信息在内存中，所有 master 的操作很快。另外，master 可以高效地定期在后台 scan 所有的 meta 数据，来执行垃圾回收、副本修复、均衡等。metadata 都记录在内存中，所以 GFS 系统会比较关注 chunk 的数量以及 master 的可用内存量。但是在实际场景下，这不是问题。每个 64MB 的 chunk 的 metadata 小于 64 字节，大部分的 chunk 都是满负荷存储的，除了文件最后一个 chunk 的空间是没有完全被占用。由于文件的名字空间采用了前缀压缩的方式存储，单个文件的 meta 信息也是小于 64 字节。如果需要扩大系统规模的话，可以很简单地通过增大 master 的内存就可以了。相比于系统的高可靠、高性能和简洁性，增加内存是很最小的代价了。

5.2 chunk 分布

并没有持久化存储 chunk 的副本分布信息，而是在 master 启动时向 chunkserver 查询其 chunk 信息，然后通过 heartbeat 来持续更新 master 的副本分布信息，以与 chunkserver 数据保持一致。GFS 起初设计时尝试将 chunk 的分布信息持久化存储在 master 端，随后发现通过 master 启动时拉取然后通过 heartbeat 同步 chunk 信息的方式更简单。因为，当 chunkserver 加入、退出、名字改变、重启等行为经常发生，这会导致维护 master 的 chunk meta 数据的正确性是非常困难的。从另一个角度考虑就是，只有 chunkserver 汇报的 chunk 信息才是集群中最真实的 chunk 分布，因为 master 不需要自己维护一个 chunk 分布状态，只需要以 chunkserver 的状态汇报为准即可。

5.3 操作日志

日志记录了 GFS 集群数据更改的历史记录。操作日志对 GFS 来说是至关重要的，因为它不仅是 metadata 的持久化记录，还记录了并发操作的时序。因为操作日志很重要，所以必须可靠地存储。在 metadata 的 change 没有持久化之前，client 是不能看到的数据的更改。当 client 修改数据时，操作记录需要保存在多个远端机器上，而且只有当操作记录持久化存储在本地和远端以后，才会回复 client 数据更改成功。

可以通过回放操作日志来恢复文件系统。为了减少系统启动时 replay 的时间，必须缩减回放的日志量。master 可以定期存储 metadata 的 checkpoint，master 重启时可以从 checkpoint 加载 metadata，然后回放 checkpoint 之后的少量日志即可。

6 总结

Google 的成功表明单 master 的设计师可行的。这不仅简化了系统，而且能够较好地实现一致性，给予性能考虑，GFS 提出了 “记录至少原子性追加一次” 的一致性模型。通过租约的方式将每个 chunk 的修改授权到 chunkserver 从而减少了 master 的负载，通过流水线的方式复制多个副本以减少延时。master

维护的元数据很多，需要设计高效的数据结构，且要保证占用内存小和支持快照操作。支持 COW 的 B 树可以满足需求，但是实现确实相当复杂。