# The Google File System

SOSP’03, October 19–22, 2003, Bolton Landing, New York, USA

GFS的实现和论文发表，是一个经典里程碑，为分布式计算时代的到来奠定了架构基础，建议结合MapReduce深入研读。

## 摘要：

GFS是面向大规模数据密集型应用的、可伸缩的分布式文件系统。运行在廉价的普遍硬件设备上，但依然具备容灾能力，为大量客户机提供高聚合性能服务。

GFS 和早期的分布式文件系统的设计思路有明显不同（因为GFS服务于Google自身业务，设计理念由Google的应用负载特征决定）。

## 1 简介：

除了传统分布式文件系统设计的目标，如性能、扩展性、高可靠和高可用，GFS进行了新的设计，因为要考虑实际应用。（对应摘要）

一、大型系统中，组件失效是常态事件。

二、应用决定了文件属于大文件。（因此很多参数和处理方法需要调整）

三、应用决定很多文件写操作属于追加写。（Append模式）写完之后，通常只有读，而且是顺序读。

四、应用程序和文件系统 API 协同设计，提高系统灵活性（双方配合）。对一致性要求不需要完全符合传统文件系统要求。

## 2 设计概要：

**2.1 Assumptions假设**

同简介中四特性！

多客户端对同一文件进行追加写，用于“生产者-消费者”队列，或其它多路文件合并操作。必须用最小同步开销实现原子性多路追加写操作。

此外，对网络要求：高性能和稳定的重要性大于低延迟。（网络性能的两个维度的指标）

**2.2 接口**

GFS 提供了一套类似传统文件系统的 API 接口，但不严格遵照POSIX语义。（建议查看POSIX语义的概念）。分层目录结构、路径名标识（C语言中文件读写中有此类概念，操作系统中也介绍过），支持create, delete,open, close, read, and write等操作。

GFS 提供了快照和记录追加操作。快照（某一特定时刻的镜像，也称为即时拷贝）以很低的开销创建一个文件或者目录树的拷贝。记录追加操作方式，可以支持多个客户端同时对一个文件进行数据追加操作，同时保证每个客户端的追加操作都是原子性的。这对于实现多路结果合并，以及“生产者-消费者”队列非常有用，多个客户端可以在不需要额外的同步锁定的情况下，同时对一个文件追加数据。

**2.3 架构**

一个GFS集群包含一个单独的Master节点（主控节点）、多台Chunk（数据块）服务器。GFS同时被多个客户端访问，如图1所示。所有的这些机器可由普通的、廉价的Linux服务器组成，运行用户级别(user-level)的服务进程。（注意：这里就是我们介绍过的横向扩展概念，不需要昂贵复杂的高端设备，而是由同质化、相对廉价的标准通用服务器构成分布式集群。同时，以用户态运行，具备比内核态实现复杂度低等特点）

GFS存储的文件都被分割成固定大小的Chunk。创建Chunk时，Master节点给每个Chunk分配一个不变的、全球唯一的64位的Chunk标识。Chunk服务器以普通Linux 文件的形式将Chunk保存在本地硬盘，并且根据指定的Chunk标识和字节范围来读写块数据。为提高数据可靠性，每个块都会复制到多个块服务器上进行备份。缺省情况下，使用三份备份（通常称为副本），备份策略可由用户指定。 Master 节点管理文件系统的元数据（metadata，不同于通常意义的数据data）。这些元数据包括命名空间（namespace）、访问控制信息、文件和Chunk的映射信息、以及当前 Chunk 的位置信息（注意，GFS集群中用户的文件，被分块存放于本地文件中，所以只有知道文件分块信息、分块存放位置的信息，才能对文件进行读写）。Master 节点还管理着系统范围内的活动，比如，Chunk 租用管理、孤儿 Chunk的回收、以及 Chunk 在 Chunk 服务器之间的迁移等。Master节点使用心跳（注意：heartbeat，以固定的频率向其他节点汇报当前节点状态的方式。用于保持联络、侦测对方节点状态的机制，多用于HA策略）信息周期地和每个 Chunk服务器通讯，发送指令到各个 Chunk 服务器并接收 Chunk 服务器的状态信息。

GFS客户端代码以库的形式被链接到客户程序里。客户端代码实现了GFS文件系统的API接口函数。客户端代表应用程序与 Master节点和Chunk 服务器通讯、以便对数据进行读写操作。客户端和 Master 节点的通信只获取元数据，所有的数据操作都是由客户端直接和 Chunk 服务器进行交互的。我们不提供 POSIX 标准的 API 的功能，因此，GFS API调用不需要深入到 Linux vnode 级别。（注意！这里提到了客户端的工作方式，实现在用户态，不需要深入内核。提到了带外模式：通Master通信只获取元数据，数据由客户端直接与Chunk服务器交互，这样分解了Master的工作负载。提到了不遵循posix语义，也就是说并非通常意义标准文件系统，这也是为Google应用定制的）

客户端和Chunk服务器都不需要缓存文件数据。

**2.4 单一Master节点**

再次提到带外模式（客户端并不通过Master节点读写文件数据。客户端向Master节点询问它应该联系的Chunk 服务器。客户端将这些元数据信息缓存一段时间，后续的操作将直接和Chunk 服务器进行数据读写操作。），着重了解元数据、数据的关系，读数据的流程。

描述读操作的流程。可结合之前MapReduce工作流程中数据读取环节理解。

还涉及到了缓存和预读的概念。

**2.5 Chunk 尺寸**

Chunk尺寸的概念，需要预先理解文件系统Block size概念。

由于实际应用中通常是大文件，所以可以选用较大的Chunk尺寸，而这可以从多个角度减少系统负载。（注意：当然也会带来负面因素。这是一个权衡的过程，综合考虑实际负载特征。）

**2.6 元数据**

Master服务器存储3种主要类型的元数据，包括：文件和Chunk的命名空间、文件和Chunk的对应关系、每个Chunk副本的存放地点。 所有的元数据都保存在 Master 服务器的内存中。前两种类型的元数据同时也会以记录变更日志的方式记录在操作系统的系统日志文件中，日志文件存储在本地磁盘上，同时日志被复制到其它的远程Master服务器上。 采用保存变更日志的方式，我们能够简单可靠的更新 Master 服务器的状态，并且不用担心Master服务器崩溃导致数据不一致的风险。Master服务器不持久保存Chunk位置信息。Master服务器在启动时，或者有新的Chunk服务器加入时，向各个Chunk服务器轮询它们所存储的Chunk信息。

2.6.1 内存中的数据结构

元数据保存在Master内存中，所以Master 操作速度非常快。即便支持规模庞大的文件系统，Master服务器内存开销也较少。

2.6.2 Chunk 位置信息

如前所述，Master服务器不持久保存Chunk的位置信息。启动的时候轮询 Chunk 服务器，之后定期轮询更新。通过Chunk 服务器确定一个Chunk是否在其上存储。不用Master服务器维护这些信息的全局视图

2.6.3 操作日志

操作日志包含了关键的元数据变更历史记录。这对 GFS非常重要（元数据对于文件系统的重要性不言自明）。操作日志是元数据唯一的持久化存储记录，也记录了同步操作顺序的逻辑时间线。（操作日志非常重要！如果出错，可能会影响整个系统正确性。）把日志复制到多台远程机器，并且只有把相应的日志记录写入到本地以及远程机器的硬盘后，才会响应客户端的操作请求。

Master 服务器在灾难恢复时，通过重演操作日志把文件系统恢复到最近的状态。（注意：提到了Checkpoint的概念，类似于快照。）

**2.7 一致性模型**

GFS的一致性要求相对传统文件系统较低（应用特征决定的），因此简单、便于实现，并且提高效率。这部分内容难度较大，大家可以通过前期视频课程（<https://www.bilibili.com/video/BV1wK4y1s7ph/?p=1>， 第37分钟开始），有个感性认识即可。

建议了解清楚基本概念和基本的一致性保障思路与方法，体会追加写对并发处理和一致性维护的影响。例如：2.7.1 GFS一致性保障机制中，提到：文件命名空间的修改（例如，文件创建）是原子性的。仅由Master节点控制：命名空间锁提供了原子性和正确性（4.1 章）的保障；Master节点的操作日志定义了这些操作在全局的顺序（2.6.3 章）。同时，2.7.2中特意提到，上述一致性模型需要应用程序的配合，例如通过追加写方式。

## 3 系统交互

系统设计的重要原则：最小化所有操作与Master节点的交互（单点瓶颈！）。

该部分描述客户机、Master服务器、Chunk服务器的交互方式，如何实现数据修改操作、原子的记录追加操作以及快照功能

**3.1 租约（lease）和变更顺序 （个人理解 – 实际是个缓存的概念）**

使用租约（lease）机制来保持多个副本间变更顺序的一致性。建立租约就是“临时下放”Master的管理权限给某一个Chunk，也就是主chunk（primary），其他副本都遵照主Chunk命令执行。”（注意：有点类似分工或者权力下放，这样每个任务都分配给一个主Chunk负责，Master的压力就很小了。主Chunk负责让其他副本（称为二级副本，secondary）存放服务器更新副本。）租约机制极大减小了Master节点的管理负担。

当然其中机制还是比较复杂（比如心跳、主Chunk延长租约请求、Master提前收回租约等），弄清楚基本原理即可。

注意一点，其中提到主Chunk对Chunk的所有更改操作进行序列化，所有的副本都遵从这个序列进行修改操作。而客户机把数据推送到所有的副本上。

时间关系没法展开介绍，其实这段逻辑非常严谨，关联性很强。

**3.2 数据流**

数据操作和元数据操作分离，各走各的路，减少Master负载。这就是带外模式（out-of band,与带内模式对应）。而Chunk服务器更新副本的过程，采用了优化策略进行数据推送（例如，尽量在网络拓扑中选择还没有接收到数据的、离自己最近的机器作为目标节点推送数据）。

时间关系没法展开介绍，其实这段逻辑非常严谨，关联性很强。

**3.3 原子记录追加**

不同于传统写操作，GFS 提供了一种原子的数据追加操作–记录追加。（注意：之所以采用这种方式，是由于，在分布式系统中，经常有很多的客户端并行地对同一个文件写数据。如果采用传统方式写文件，分布式系统需要额外的复杂、昂贵的同步机制，而类似O\_APPEND 模式的追加写，并发性不需要竞争。写操作、副本更新操作需要提供一致性保障。）

时间关系没法展开介绍，其实这段逻辑非常严谨，关联性很强。

**3.4 快照**

快照操作几乎可以瞬间完成对一个文件或者目录树（“源”）做一个拷贝，并且几乎不会对正在进行的其它操作造成任何干扰。可以使用快照迅速的创建一个巨大的数据集的分支拷贝，或者是在做实验性的数据操作之前，使用快照操作备份当前状态，这样之后就可以轻松的提交或者回滚到备份时的状态。利用copy-on-write技术实现快照。（注意：相当于系统每一步操作都被记录、备份下来。类似于游戏中的存档、读档）

时间关系没法展开介绍，其实这段逻辑非常严谨，关联性很强。

## 4 Master节点操作

Master节点执行所有的名称命名操作。此外，它还管理着整个系统里所有 Chunk的副本：它决定Chunk的存储位置，创建新Chunk和它的副本，协调各种各样的系统活动以保证Chunk被完全复制，在所有的Chunk服务器之间的负载均衡，回收不再使用的存储空间。

**4.1 命名空间（namespace）管理和锁**

采用锁的方式保障同一目录的并行操作。

**4.2 副本的位置**

Chunk副本位置选择的策略服务两大目标：最大化数据可靠性和可用性，最大化网络带宽利用率。

**4.3 创建，重新复制，重新负载均衡**

思路清晰、简单。比如分配副本时，优先选取在存储资源利用率低的服务器；Chunk的副本分布在多个机架间等。

而当Chunk有效副本数量少于用户指定的复制因数的时候，Master 节点会重新复制。比如某Chunk服务器故障或者磁盘损坏。

**4.4 垃圾回收**

惰性回收策略

**4.5 过期失效的副本检测**

## 5 容错和诊断

（注意，之前我们提到，大型分布式系统，组件失效视为常态，因此对容错的处理和诊断就是必须考虑的。）不能仅依赖机器的稳定性和硬盘的可靠性。组件的失效可能造成系统不可用，甚至可能产生不完整的数据。

**5.1 高可用性（保证系统正常提供服务）**

两个策略：快速恢复和复制

**5.1.1 快速恢复**

不管Master服务器和Chunk服务器正常关闭或异常关闭，可以在数秒钟内恢复并重新启动。

**5.1.2 Chunk 复制**

当有Chunk服务器离线或者通过Chksum 校验发现了已经损坏的数据，Master节点通过克隆已有的副本保证每个 Chunk 都被完整复制。（注意，副本机制虽然简单有效，但是额外的存储空间占用实在是太大了。文中也提到了其它形式的跨服务器的冗余解决方案，比如使用奇偶校验、或者Erasure codes。其中EC是非常成功的数据冗余校验技术，纠删码。当然，在了解纠删码技术之前，大家应该先掌握RAID技术工作原理。）

**5.1.3 Master服务器的复制**

为了保证 Master 务器的可靠性，Master服务器的状态也要复制。Master服务器所有的操作日志和checkpoint文件都被复制到多台机器上。

此外，GFS 中还有些“shadow”Master 服务器，Master 服务器宕机的时候提供文件系统的只读访问。

**5.2 数据完整性（或称数据可靠性，注意和“高可用”概念的区别）**

每个 Chunk 服务器都使用Checksum来检查保存的数据是否损坏。GFS 允许有歧义的副本存在。

对于读操作来说，在把数据返回给客户端或者其它的 Chunk 服务器之前，Chunk 服务器会校验读取操作涉及的范围内的块的 Checksum。因此 Chunk 服务器不会把错误数据传递到其它的机器上。如果发生某个块的Checksum 不正确，Chunk 服务器返回给请求者一个错误信息，并且通知 Master 服务器这个错误。作为回应，请求者应当从其它副本读取数据，Master 服务器也会从其它副本克隆数据进行恢复。当一个新的副本就绪后，Master 服务器通知副本错误的 Chunk 服务器删掉错误的副本。

在 Chunk 服务器空闲的时候，它会扫描和校验每个不活动的 Chunk 的内容。

**5.3 诊断工具**

尽量保存系统日志

（注意，提到了RPC日志，大家可以了解一下RPC的概念和工作原理）

## 6 测试

用基准测试用例和实际应用数据进行测试，分为读操作、写操作、追加写操作（Append）。

注意：

测试时要考虑内存的影响，区分读写比例、读写操作性能不同

注意Master服务器负载的分析，因为Master是单点，但是测试显示Master不是系统瓶颈。

故障恢复测试

工作负载的分析

最后：通过本文的测试，希望大家可以真正思考，测试的目的、测试的方法，从测试数据分析中可以获得有效信息、发现问题进而对改进指出方向。相比之下，我们毕设论文有的只展现了实现系统的几个功能截图，是非常不充分甚至没有意义的。

## 7 经验总结

工作里遇到的一些困难与解决思路和方案

## 8 相关工作

类似我们论文中的国内外研究现状，清楚对比了GFS与其他文件系统的异同。

## 9 结束语

总结和后期工作预期展望

## 10 致谢

略