**GFS阅读报告**

1. 在探索设计空间中得到的一些观点。

第一：组建故障是常态而不是例外。这个系统由许多廉价易损的普通组件组成。它必须持续监视自己的状态，它必须在组件失效作为一种常态的情况下，迅速地侦测，承担并恢复那些组件失效。

第二：以传统标准来看，文件非常大。预期有几百万文件，大小通常是100MB或者以上。数GB的文件也很寻常，而被有效的管理。小文件必须支持，但是不需要去优化。

第三：大多数文件通过附加新数据而不是通过覆盖现有的数据而发生变化。文件中的随机写操作是不存在的。一旦写入，就只能读取文件，而且就只能按顺序读取。

第四：协同设计应用程序和文件系统API可以提高灵活性，从而使整个系统收益。

1. 假设。

* 该系统是由许多经常失败的廉价商品组件构建的。它必须经常监测自己，发现，容忍从组件故障中及时恢复。
* 系统储存的大文件数量适中。
* 工作负载主要由两种读取组成：大型流媒体读取和小型随机读取。
* 工作负载也有许多大的，连续的写操作，他们将数据追加到文件中。
* 系统必须有效地为多个客户端实现定义良好的语义，同时追加到同一文件。
* 高持续带宽比低延迟更重要。大多数我们的目标程序，在高速率传输下，大块地操作数据，因为大部分单独的读写操作没有严格的响应时间要求。

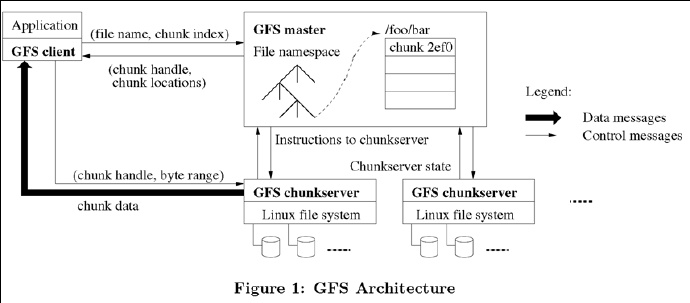
1. 接口。

GFS提供了一套类似传统文件系统的API接口，虽然并不是严格按照POSIX(The Portable Operating System Interface)等标准实现的。

* **快照：快照以很低的成本创建一个文件或者目录树的拷贝。**
* **记录追加：操作允许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。**

1. 体系结构。

GFS主要由以下三个系统模块组成：

* Master：管理元数据，整理协调系统活动。
* ChunkServer：储存维护数据块（Chunk），读写文件数据。
* Client：向Master请求元数据，并根据元数据访问对应ChunkServer的Chunk。
* 

如图所示：首先，使用固定的块大小，客户端将应用程序指定的文件名和字节偏移量转换为文件中的块索引。然后，它向master发送一个包含文件名和块索引的请求。主服务器使用相应的块语句柄和副本的位置进行应答。客户端使用文件名和块索引作为键来缓存这些信息。然后客户端向其中一个副本（最有可能是最近的副本）发送请求。请求指定块句柄和该块中的字节范围。在缓存的信息过期或文件重新打开之前，对同一块的进一步读取不需要更多的客户-主服务器交互。客户端通常在同一个请求中请求多个块，而主服务器也可以在被请求的块之后立即包含这些块的信息。这些额外的信息几乎不需要额外的成本就可以避开未来的几个客户机-主机交互。

1. 块大小。

* Chunk 的大小是关键的设计参数之一。GFS选择了**64MB**，这个尺寸远远大于一般文件系统的 Block size。每个 Chunk 的副本都以普通 Linux 文件的形式保存在 Chunk 服务器上，只有在需要的时候才扩大。这种chunk尺寸可以减少对master节点的请求；可以保持长的TCP请求；减少元数据同时缺点也非常明显，小文件只有一个chunk的话可能会导致热点问题
* 块大小的优点：

第一：他减少了客户机与主服务器交互的需要，因为对同一块进行读写只需要向服务器发出一次初始请求，以获取块位置信息。

第二：由于在一个块的大小上，客户端更有可能对给定的块执行许多操作，他可以通过保持persistent来减少开销，延长与chunkserver的TCP连接时间。

第三：他减少了储存在主服务器上元数据的大小，允许将元数据保存在内存中。

1. 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，同时日志会被复制到其它的远程 Master服务器上。

1. 一致性模型。

GFS又一个宽松的一致性模型，它可以很好的支持高度分布式的应用程序，但是实现起来任然相对简单和高效。

表格

描述已自动生成

1. 系统交互。

**原则：最小化所有操作和Master节点的交互。**

* 使用租约lease机制来保持多个副本间变更顺序的一致性

图示

描述已自动生成

* 数据流。

沿着Chunk服务器数据链进行推送，接受数据的Chunk服务器同时通过管道在全全双工的模式下向距离自己最近的同时也在链路上的Chunk服务器推送。在没有网络阻塞的情况下，传送B字节到R个副本的理想时间B/T+L\*R，T是吞吐量，L是两台机器之间的传输延迟。

* 原子记录增加

GFS 提供了一种原子的数据追加操作–记录追加。传统方式的写入操作，客户程序会指定数据写入的偏移量。对同一个 region 的并行写入操作不是串行的： region 尾部可能会包含多个不同客户机写入的数据片段。使用记录追加，客户机只需要指定要写入的数据。 GFS 保证至少有一次原子的写入操作成功执行（即写入一个顺序的 byte 流），写入的数据追加到 GFS 指定的偏移位置上，之后 GFS 返回这个偏移量给客户机。这类似于在 Unix 操作系统编程环境中，对以 O\_APPEND 模式打开的文件，多个并发写操作在没有竞态条件时的行为。

* 快照

使用copy-on-write技术实现快照。

当 Master 节点收到一个快照请求，它首先取消作快照的文件的所有 Chunk 的租约。这个措施保证了后续对这些 Chunk 的写操作都必须与 Master 交互以找到租约持有者。这就给 Master 节点一个率先创建Chunk 的新拷贝的机会。

1. **Master节点的操作。**

* **名称空间和锁。**

**GFS不希望在一些工作量较大的master节点的操作的运行时，延缓了其它的 Master 节点的操作。因此，允许多个操作同时进行，使用名称空间的 region 上的锁来保证执行的正确顺序。 因为名称空间可能有很多节点，读写锁采用惰性分配策略，在不再使用的时候立刻被删除。同样，锁的获取也要依据一个全局一致的顺序来避免死锁：首先按名称空间的层次排序，在同一个层次内按字典顺序排序。**

* **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。**

* **创建，重新副本，重新负载均衡。**

**需要考虑的几个因素：**

* + **平衡硬盘使用率。**
  + **限制同一台Chunk服务器上进行的创建/复制的操作数量。**
  + **在机架间分布副本。**
* **垃圾回收。**

**一开始只做日志标记，将文件隐藏，然后过几天在做删除，所有不能被Master节点识别的副本将被删除。垃圾回收会分散到各种例行操作里，同时会在Master节点相对空闲的时候完成。master节点会保存副本的版本号确保访问数据的正确性。**

1. 容错和诊断。

* **高可用性：**
* **快速恢复：不管 Master 服务器和 Chunk 服务器是如何关闭的，它们都被设计为可以在数秒钟内恢复它们的状态并重新启动。**
* **Chunk复制：虽然 Chunk复制策略对我们非常有效，但是我们也在寻找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来解决我们日益增长的只读存储需求。**
* Master的复制：对 Master 服务器状态的修改操作能够提交成功的前提是，操作日志写入到 Master 服务器的备节点和本机的磁盘。 GFS 中还有些“影子” Master 服务器，这些“影子”服务器在“主” Master 服务器宕机的时候提供文件系统的只读访问。这里要注意，我们都知道Namenode在Master上，但是SecondaryNamenode并不是其备份，而是辅助Namenode工作的。并不是这里提到的备用节点
* 数据完整性。

**每个块服务器使用校验和来检测存储数据的损坏。一块被分成64 KB的块。每个都有对应的32位校验和。像其他元数据一样，校验和与用户数据分开保存在内存中，并通过日志记录持久存储。对于读取，块服务器在将任何数据返回给请求者（无论是客户端还是其他块服务器）之前，会验证与读取范围重叠的数据块的校验和。因此，块服务器将不会将损坏传播到其他计算机。**

**由于以下几个原因，校验和对读取性能的影响很小：**

* 由于我们的大多数读取操作至少跨越几个块，因此我们仅需要读取和校验和相对少量的额外数据即可进行验证。 GFS客户端代码通过尝试在校验和块边界对齐读取来进一步减少了这种开销。
* 此外，无需任何I / O即可完成对块服务器的校验和查找和比较，并且校验和计算通常会与I / O重叠。

**由于校验和计算在我们的工作负载中占主导地位，因此校验和计算针对附加到块末尾的写入（与覆盖现有数据的写入相反）进行了优化。我们只是增量地更新最后的部分校验和块的校验和，并为附加的填充的任何全新校验和块计算新的校验和。在空闲期间，块服务器可以扫描并验证非活动块的内容。这使我们能够检测很少读取的块中的损坏。一旦检测到损坏，主服务器即可创建新的未损坏副本，并删除损坏的副本。这样可以防止不活动但已损坏的块副本欺骗主服务器，使其认为它具有足够的有效块副本。**

* **诊断工具。**

**广泛而详细的诊断日志记录在问题隔离，调试和性能分析方面提供了不可估量的帮助，同时仅产生了最小的成本。如果没有日志，则很难理解机器之间的瞬时，不可重复的交互。 GFS服务器生成诊断日志，该日志记录了许多重要事件（例如，上下启动的块服务器）以及所有RPC请求和答复。这些诊断日志可以自由删除，而不会影响系统的正确性。但是，尝试将这些日志保留在空间允许的范围内。**

**RPC日志包括在线上发送的确切请求和响应，但读取或写入的文件数据除外。通过将请求与答复进行匹配并在不同的计算机上整理RPC记录，可以重建整个交互历史以诊断问题。日志还可用作负载测试和性能分析的跟踪。**

**日志记录对性能的影响很小（远远超过了好处），因为这些日志是顺序和异步写入的。最新事件也保存在内存中，可用于连续在线监视。**

1. **测量。**

* **微型基准测试。**

**在由一个主服务器，两个主副本，16个块服务器和16个客户端组成的GFS群集上测量了性能。此配置是为了便于测试而设置的。 典型的集群具有数百个块服务器和数百个客户端。**

**所有机器都配置有双1.4 GHz PIII处理器，2 GB内存，两个80 GB 5400 rpm磁盘以及与HP 2524交换机的100 Mbps全双工以太网连接。 所有19台GFS服务器计算机都连接到一台交换机，所有16台客户机计算机都连接到另一台。 两个交换机通过1 Gbps链路连接。**

* **读取。**

**N个客户端同时从文件系统读取。每个客户端从320 GB的文件集中读取随机选择的4 MB区域。重复256次，以便每个客户端最终读取1 GB数据。合在一起的块服务器仅具有32 GB的内存，因此期望Linux缓冲区高速缓存中的命中率最高为10％。结果应该接近冷缓存结果。**

**图3（a）显示了N个客户端的总读取速率及其理论极限。当两个交换机之间的1 Gbps链路达到饱和时，限制的峰值总计为125 MB / s；如果其客户端的100 Mbps网络接口达到饱和，则每个客户端为12.5 MB / s（以适用者为准）。当仅一个客户端正在读取时，观察到的读取速率为10 MB / s，即每个客户端限制的80％。对于16个读取器，总读取速率达到94 MB / s，约为125 MB / s链接限制的75％，或每个客户端6 MB / s。效率从80％下降到75％，因为随着读取器数量的增加，多个读取器同时从同一块服务器读取数据的可能性也随之增加。**

图示

描述已自动生成

* **写。**

**N个客户端同时写入N个不同的文件。每个客户端以1 MB的写入顺序将1 GB的数据写入新文件。总写入速率及其理论极限如图3（b）所示。极限平稳期为67 MB / s，因为需要将每个字节写入16个块服务器中的3个，每个块服务器具有12.5 MB / s的输入连接。**

**一个客户端的写入速率为6.3 MB / s，约为限制的一半。造成这种情况的主要原因是我们的网络堆栈。它与我们用于将数据推送到大块副本的流水线方案互动性不是很好。将数据从一个副本传播到另一个副本的延迟会降低总体写入速率。**

**16个客户端的总写入速率达到35 MB / s（每个客户端2.2 MB / s），约为理论上限的一半。与读取的情况一样，随着客户端数量的增加，多个客户端并发写入同一块服务器的可能性更高。而且，与每个16个读写器相比，与16个读写器发生冲突的可能性更大，因为每个写入涉及三个不同的副本。**

**写的速度比我们想要的慢。实际上，这并不是一个主要问题，因为即使它增加了各个客户端看到的延迟，也不会显着影响系统交付给大量客户端的聚合写带宽。**

* + **追加。**

**图3（c）显示了记录追加性能。 N个客户端同时附加到单个文件。 性能由存储文件的最后一个块的块服务器的网络带宽限制，而与客户端数量无关。 对于一个客户端，它的开始速度为6.0 MB / s，对于16个客户端，它的速度降至4.8 MB / s，这主要是由于不同客户端看到的网络传输速率的拥塞和差异所致。目前，它的应用程序倾向于生成多个此类文件。 换句话说，N个客户端同时附加到M个共享文件，其中N和M都在数十个或数百个中。 因此，在的实验中，chunkserver网络拥塞实际上并不是一个重要问题，因为客户端可以在编写一个文件时取得进展，而另一文件的chunkserver却很忙。**

* + **现实世界中集群。**

**现在，检查一下Google内部正在使用的两个集群，它们分别代表了其他几个集群。**

**数百名工程师定期将Cluster A用于研发。 一个典型的任务是由人类用户发起的，并且运行时间长达几个小时。 它读取从几MB到几TB的数据，转换或分析数据，然后将结果写回群集。**

**群集B主要用于生产数据处理。 这些任务的持续时间更长，并且仅在偶尔的人工干预下就可以连续生成和处理多结核病数据集。 在这两种情况下，单个“任务”都由许多计算机上的多个进程同时读取和写入许多文件组成。**

* + **存储。**

**如表中的前五个条目所示，两个集群都有数百个块服务器，支持许多TB的磁盘空间，并且相当但不是完全满。 “已用空间”包括所有块副本。几乎所有文件都被复制3次。因此，群集分别存储18 TB和52 TB的文件数据。**

**这两个群集具有相似数量的文件，尽管B具有更多的失效文件，即已删除或替换为新版本但尚未回收其存储的文件。它还具有更多的块，因为其文件往往更大。**

表格

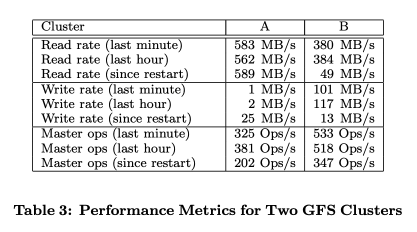
描述已自动生成

### 读写速率。

**表3列出了不同时间段的读写速率。 进行这些测量时，两个群集都已经运行了大约一周。 （群集最近已重新启动，以升级到新版本的GFS。）**

**自重启以来，平均写入速率小于30 MB / s。 当我们进行这些测量时，B处于写入活动突发的中间，该活动产生大约100 MB / s的数据，由于写入传播到三个副本，因此产生了300 MB / s的网络负载。**

**读取速率远高于写入速率。 如所假设的，总的工作量包含的读取次数多于写入次数。 两个集群都处于大量读活动中。 特别是，A在前一周一直保持580 MB / s的读取速率。 它的网络配置可以支持750 MB / s，因此它可以有效地利用其资源。 群集B可以支持1300 MB / s的峰值读取速率，但是其应用程序仅使用380 MB / s。**



* 主负载。

**表3还显示发送到主服务器的操作速率约为每秒200到500个操作。主机可以轻松地跟上该速率，因此对于这些工作负载而言，这不是瓶颈。**

**在早期版本的GFS中，主服务器有时是某些工作负载的瓶颈。它花费了大部分时间依次浏览大型目录（包含数十万个文件）以查找特定文件。此后，更改了主数据结构，以允许通过名称空间进行有效的二进制搜索。现在，它可以轻松地支持每秒数千个文件访问。如有必要，可以通过在名称空间数据结构之前放置名称查找缓存来进一步加快速度。**

* 恢复时间。

**块服务器发生故障后，某些块将变得复制不足，必须将其克隆以恢复其复制级别。恢复所有此类块所需的时间取决于资源量。在一个实验中，杀死了群集B中的单个块服务器。该块服务器有大约15,000个块，其中包含600 GB的数据。为了限制对正在运行的应用程序的影响并为计划决策提供余地，默认参数将该群集限制为91个并发克隆（占块服务器数量的40％），其中每个克隆操作最多允许消耗6.25 MB / s（50 Mbps）。所有块均在23.2分钟内恢复，有效复制速率为440 MB / s。**

**在另一个实验中，杀死了两个带有大约16,000个块和660 GB数据的块服务器。此双重故障将266个块减少为只有一个副本。这266个块以较高的优先级进行克隆，并在2分钟内全部恢复到至少2倍的复制，从而使群集处于可以忍受另一个块服务器故障而不会丢失数据的状态。**

* + **方法和注意事项。**

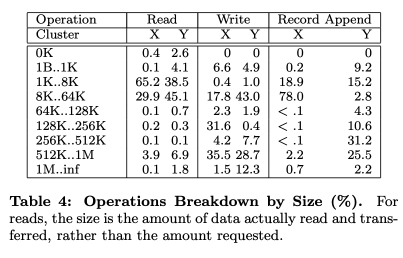
**这些结果仅包括客户端发起的请求，因此它们反映了应用程序为整个文件系统生成的工作量。它们不包括执行客户端请求的服务器间请求或内部后台活动，例如转发的写入或重新平衡。**

**I / O操作的统计信息基于从GFS服务器记录的实际RPC请求中启发式重建的信息。例如，GFS客户端代码可能将读取分为多个RPC，以增加并行度，从中推断出原始读取。由于访问模式是高度程式化的，因此希望任何错误都可能来自噪声。应用程序的显式日志记录可能会提供稍微更准确的数据，但是从逻辑上讲，重新编译并重新启动成千上万个正在运行的客户端是不可能的，而且从许多机器中收集结果很麻烦。**

**应该注意不要过度概括工作量。由于Google完全控制GFS及其应用程序，因此这些应用程序倾向于针对GFS进行调整，相反，GFS是为这些应用程序设计的。这样的相互影响也可能在通用应用程序和文件系统之间存在，但在案例中，这种影响可能会更加明显。**

* + **Chunkserver工作量。**

**表4显示了按大小划分的操作分布。读取大小显示出双峰分布。小读操作（不到64 KB）来自于搜索密集型客户端，这些客户端在大文件中查找小数据。大量读取（超过512 KB）来自对整个文件的长时间顺序读取。**

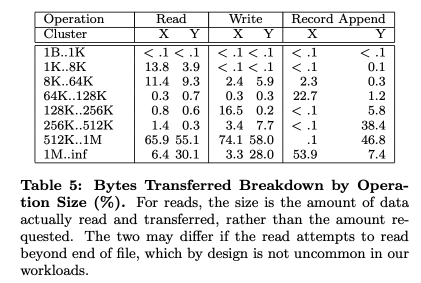


**在集群Y中，大量读取根本不返回任何数据。应用程序，尤其是生产系统中的应用程序，经常使用文件作为生产者-消费者队列。当使用者读取文件末尾时，生产者会同时将其附加到文件中。有时，当消费者超过生产者时，不会返回任何数据。群集X较少显示此信息，因为它通常用于短期数据分析任务，而不是长期分布式应用程序。**

**写入大小也表现出双峰分布。较大的写入（超过256 KB）通常是由于写入器内有大量缓冲造成的。写入器缓冲较少的数据，更频繁地检查点或进行同步，或者仅为较小的写入（小于64 KB）生成较少的数据。**

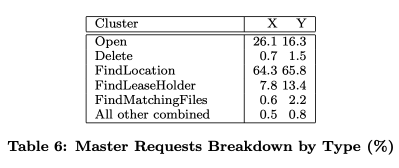
**至于记录追加，群集 Y 的大型记录追加百分比比群集 X 高得多，因为使用群集Y的生产系统针对GFS进行了更积极的调整。**

**表5列出了各种大小的操作中传输的数据总量。对于所有类型的操作，较大的操作（超过256 KB）通常占传输的大多数字节。由于随机查找工作负载，小读（64 KB以下）确实会传输一小部分但很大一部分的读数据。**



* **主工作负载。**

**表6显示了按主请求类型分类的细分。大多数请求要求块位置（FindLocation）进行读取，并要求租约所有者信息（FindLeaseLocker）进行数据变化。**



**集群X和Y看到的删除请求数量明显不同，因为集群Y存储的生产数据集会定期重新生成并替换为较新的版本。这种差异中的某些差异进一步隐藏在Open请求中的差异中，因为文件的旧版本可能会通过从头开始进行写操作而被隐式删除（Unix开放术语中的模式“ w”）。**

**FindMatchingFiles是一个模式匹配请求，它支持“ ls”和类似的文件系统操作。与对主服务器的其他请求不同，它可能处理名称空间的很大一部分，因此可能很昂贵。群集Y经常看到它，因为自动数据处理任务倾向于检查文件系统的各个部分以了解全局应用程序状态。相比之下，群集X的应用程序受用户更明确的控制，并且通常提前知道所有需要的文件的名称。**

* 结论.

**GFS 展示了在商用硬件上支持大规模数据处理工作负载所必需的质量。尽管某些设计决策是特定于独特设置的，但许多决策可能适用于规模和成本意识相似的数据处理任务。**

**首先根据当前和预期的应用程序工作负载和技术环境，重新检查传统的文件系统假设。发现导致了设计领域的根本不同。将组件故障视为正常现象，而不是例外情况，针对大型文件进行优化，这些文件通常会附加到（也许同时进行），然后再读取（通常是顺序执行），并且都扩展和放松了标准文件系统接口，以改善整个系统。**

**系统通过不断监控，复制关键数据以及快速自动恢复功能来提供容错能力。块复制使能够容忍chunkserver故障。这些故障的发生率激发了一种新颖的在线修复机制，该机制定期透明地修复损坏并尽快补偿丢失的复制品。此外，使用校验和来检测磁盘或IDE子系统级别的数据损坏，鉴于系统中的磁盘数量，这种情况变得非常普遍。**

**设计为许多同时执行各种任务的读取器和写入器提供了高总吞吐量。通过将通过主机的文件系统控制与直接在块服务器和客户端之间传递的数据传输分开来实现此目的。大型块的大小和块的租用将主操作对常见操作的影响降到最低，这将权限授予数据突变中的主副本。这样就可以成为一个简单的，集中的主服务器，而不会成为瓶颈。认为，网络堆栈的改进将解除当前对单个客户端看到的写入吞吐量的限制。**

**GFS已成功满足了存储需求，并已在Google内部广泛用作研究，开发和生产数据处理的存储平台。它是能够继续创新并解决整个Web规模问题的重要工具。**