关于*The Google File System* 的一些看法

GFS，谷歌设计并实现的文件系统，从论文中我们可以看出，该文件系统的应用范围是非常有针对性的。具体来说，谷歌文件系统是为了特定的情况，即满足大规模数据密集型应用，并且以谷歌应用的负载情况和技术环境的分析为基础的。

它假定了以下客观情况的普遍存在，同时将其反映为技术要求：

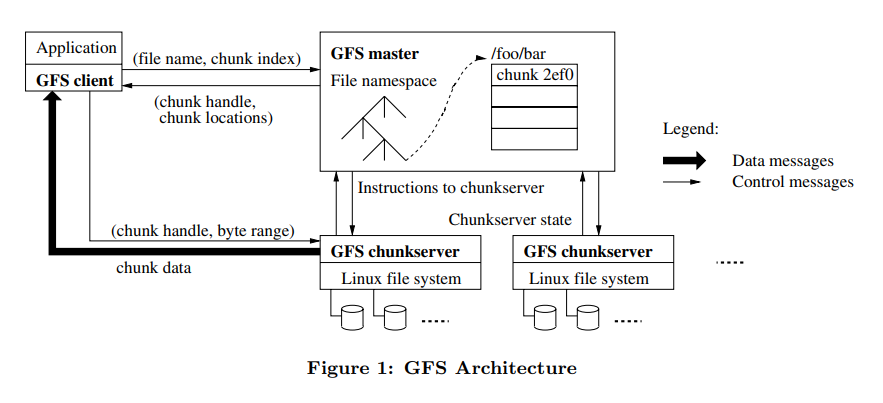
1. GFS包括几百甚至几千台普通的廉价设备组装的存储机器，因此组件失效默认为常态事件。这要求持续的监控、错误侦测、灾难冗余以及自动恢复的机制必须集成在GFS中。
2. 相比于通常的文件系统，谷歌要处理的文件非常庞大，数据集的体量并不适合多而小的管理方式。因此设计的假设条件和参数，比如I/O操作和Block的尺寸都需要重新考虑。
3. 针对海量文件的访问模式，客户端对数据块缓存是没有意义的，数据的追加操作是性能优化和原子性保证的主要考虑因素。因此绝大部分文件的修改是采用在文件尾部追加数据，而不是覆盖原有数据的方式。
4. 作为满足大规模数据密集型应用的文件系统，应用程序和文件系统API的协同设计可以提高整个系统的灵活性。因此在这一方面做出特殊的设计。例如针对高性能的稳定网络带宽进行优化和放松对一致性模型的要求。

在这些前提或者说设计原则下，一个GFS集群的架构如下所示：

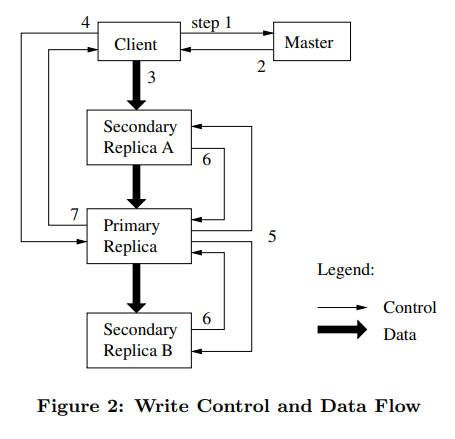
1. GFS Master（主控服务器）。管理所有的文件系统元数据。这些元数据包括名字空间、访问控制信息、文件和Chunk的映射信息、以及当前Chunk的位置信息。是一个GFS集群的核心。
2. GFS Chunk Server（CS，数据块服务器）。把文件切分成chunk分成多副本（Chunk的副本有三个用途：Chunk创建，重新复制和重新负载均衡）存储到多个chunk server上实现数据的可靠性。每个chunk有着远大于一般文件系统的block size的64MB。优点突出，但是缺点也很明显。

优点在于读写可以减少与master之间的交互，对于大块的顺序读写，这个优化是有很明显好处的。大的chunk size 使好几个操作都在同一个chunk sever的chunk可以减少TCP链接的额外消耗。缺点则集中在内部碎片问题上。

1. GFS Client（客户端）。客户端用文件名和Chunk索引作为key缓存信息，作为API库链接到具体的应用，直接通过网络访问chunk server 或者master;它从master获取数据并缓存在本地内存中，然后直接与chunk server交互，读写IO不必每次都经过master，使master不会瓶颈。



从上图我们可以看出其运转的流程：GFS存储的文件被分成了固定大小的chunk，由GFS Master也就是主控服务器分配一个全球唯一的chunk句柄，并储存三份。客户端是GFS提供给应用程序的访问接口，它是一组专用接口，以库文件的形式提供。客户端访问GFS时，首先访问主控服务器节点，获取与之进行交互的Chunk Server信息，然后直接访问Chunk Server，完成数据的存取工作。



上图展现了写入操作的控制流程：

1. 客户端向主控服务器请求chunk每个副本所在的Chunk Server。

2、主控服务器返回客户端主副本和各副本所在的Chunk Server的位置信息，客户端将缓存这些信息供以后使用。  
3、客户端将要追加的记录发送到每一个副本。GFS中采用数据流和控制流分离的方法，从而能够基于网络拓扑结构很好地调度数据流的传输。  
4、当所有副本都确认收到了数据，客户端发起一个写请求控制命令给主副本  
5、主副本把写请求提交给所有的备副本  
6、各备副本成功完成后应答主副本  
7、主副本应答客户端，如果有副本发生错误，将出现主副本写成功但某些备副本不成功的情况，客户端将重试。

**原子的记录追踪**

在系统交互中，GFS还提供了一种原子的数据追加操作——记录追加。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。

**名称空间管理和锁**

Master节点执行所有的名称空间操作，GFS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, 优点是支持对同一目录的并行操作。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。

**垃圾回收**

GFS在文件删除后不会立刻回收可用的物理空间。GFS空间回收采用惰性的策略，只在文件和Chunk级的常规垃圾收集时进行。这个方法使GPS系统更简单、更可靠。Master节点并不马上回收资源，而是把文件名改为一个包含删除时间戳的、隐藏的名字。在对Chunk名字空间做类似的常规扫描时，Master节点找到不被任何文件包含的Chunk并删除它们的元数据。而Chunk服务器可以任意删除这些Chunk的副本。这涉及到chunk server 执行垃圾回收的优点和缺点。

优点在于，由于之前创建chunk 时出现过部分成功部分失败，然后master并不知道，只有chunk自己清楚，另外delete msg 可能失败，然后master必须要记住然后重试，如果这样使设计更为复杂。同时chunk server自身扫描减轻了master的压力，同时通过心跳上报给master进行批量操作，这个消耗可以是错开分批次的，可以在master比较空闲时处理，能及时响应client的消息。但是，频繁的删除创建file，这样会导致存储空间爆满，而没有空间。当Chunk服务器失效时，Chunk的副本有可能因错失了一些修改操作而过期失效。也因此Master节点保存了每个Chunk的版本号，用来区分当前的副本和过期副本。

**容错和诊断**

之前提到由于组件的数量和质量的与众不同，组件失效问题出现的频率远远超过一般系统意外发生的频率。在这种情况下使用两条简单但是有效的策略保证整个系统的高可用性：快速恢复和复制。复制包括chunk的复制和master的复制。

个人感受：

第一次看这么长的论文。在过去两年多软件工程专业的学习中，获取知识的方式往往简单粗暴。除了老师上课所讲，无非就是搜罗些视频，试验些教程。既非是为了深化何种思想锻炼何种能力，也不图参透键盘敲击下缓慢生长的代码的真理。只想着好好完成这一份任务，完成后便心满意足（非是鼠目寸光，只是一坐便也是一天，值得“苦尽甘来”的小小快乐）。而到了今日，方知一直囿于不太舒适的舒适圈。要看下这论文，虽然刚算傍晚，我却就如中午妄图躺床上背诵英语单词一般，止不住在自习室座位上渐渐低头又重重坠醒。果然还是要多加练习啊。

GFS让我理解何为针对于某些需求而开发的特定系统，也见识到了开发系统所需要考虑的方方面面——即使他们仍仅限于书面表达的理论知识而没有涉及亲手构造的代码世界。