从 HDFS 看分布式文件系统的设计需求

分布式文件系统的设计需求大概是这么几个：透明性、并发控制、可伸缩性、容错以及安全需求等。我想试试从这几个角度去观察 HDFS 的设计和实现，可以更清楚地看出 HDFS 的应用场景和设计理念。

首先是透明性，如果按照开放分布式处理的标准确定就有 8 种透明性：访问的透明性、位置的透明性、并发透明性、复制透明性、故障透明性、移动透明性、性能透明性和伸缩透明性。对于分布式文件系统，最重要的是希望能达到 5 个透明性要求：

1）访问的透明性：用户能通过相同的操作来访问本地文件和远程文件资源。HDFS 可以做到这一点，如果 HDFS 设置成本地文件系统，而非分布式，那么读写 分布式 HDFS 的程序可以不用修改地读写本地文件 ,要做修改的是配置文件。可见，HDFS 提供的访问的透明性是不完全的，毕竟它构建于 java 之上，不能 像 NFS 或者 AFS 那样去修改 unix 内核，同时将本地文件和远程文件以一致的方式处理。

2）位置的透明性：使用单一的文件命名空间，在不改变路径名的前提下，文件或者文件集合可以被重定位。HDFS 集群只有一个 Namenode 来负责文件系 统命名空间的管理，文件的 block 可以重新分布复制，block 可以增加或者减少副本，副本可以跨机架存储，而这一切对客户端都是透明的。

3）移动的透明性，这一点与位置的透明性类似，HDFS 中的文件经常由于节点的失效、增加或者 replication 因子的改变或者重新均衡等进行着复制或者移动，而客户端和客户端程序并不需要改变什么，Namenode 的 edits 日志文件记录着这些变更。

4）性能的透明性和伸缩的透明性 :HDFS 的目标就是构建在大规模廉价机器上的分布式文件系统集群，可伸缩性毋庸置疑，至于性能可以参考它首页上的一些 benchmark。

其次是并发控制，客户端对于文件的读写不应该影响其他客户端对同一个文件的读写。要想实现近似原生文件系统的单个文件拷贝语义，分布式文件系统需要做

出复 杂的交互，例如采用时间戳，或者类似回调承诺（类似服务器到客户端的 RPC 回调，在文件更新的时候；回调有两种状态：有效或者取消。客户端通过检查回调承 诺的状态，来判断服务器上的文件是否被更新过）。HDFS 并没有这样做，它的机制非常简单，任何时间都只允许一个写的客户端，文件经创建并写入之后不再改变，它的模型是 write-one-read-many， 一次写，多次读。这与它的应用场合是一致，HDFS 的文件大小通常是兆至 T 级的，这些数据不会经常修改，**最经常的是被顺序读并处理，随机读很少，因此 HDFS 非常适合 MapReduce 框架或者 web crawler 应用。**HDFS 文件的大小也决定了它的客户端不能像某些分布式文件系统那样缓存常用到的几百个文件。

第三，文件复制功能，一个文件可以表示为其内容在不同位置的多个拷贝。这样做带来了两个好处：访问同个文件时可以从多个服务器中获取从而改善服务的伸缩 性，另外就是提高了容错能力，某个副本损坏了，仍然可以从其他服务器节点获取该文件。HDFS 文件的 block 为了容错都将被备份，根据配置的 replication 因子来，默认是 3。副本的存放策略也是很有讲究，一个放在本地机架的节点，一个放在同一机架的另一节点，另一个放在其他机架上。这 样可以最大限度地防止因故障导致的副本的丢失。不仅如此，HDFS 读文件的时候也将优先选择从同一机架乃至同一数据中心的节点上读取 block。

第四，硬件和操作系统的异构性。由于构建在 java 平台上，HDFS 的跨平台能力毋庸置疑，得益于 java 平台已经封装好的文件 IO 系统，HDFS 可以在不同的操作系统和计算机上实现同样的客户端和服务端程序。

第五，容错能力，在分布式文件系统中，尽量保证文件服务在客户端或者服务端出现问题的时候能正常使用是非常重要的。HDFS 的容错能力大概可以分为两个方面：文件系统的容错性以及 Hadoop 本身的容错能力。文件系统的容错性通过这么几个手段：

1）在 Namenode 和 Datanode 之间维持心跳检测，当由于网络故障之类的原因 导致 Datanode 发出的心跳包没有被 Namenode 正常收 到的时候，Namenode 就不会将任何新的 IO 操作派发给那个 Datanode，该 Datanode 上的数据被认为是无效的，因此 Namenode 会检 测是否有文

件 block 的副本数目小于设置值，如果小于就自动开始复制新的副本并分发到其他 Datanode 节点。

2）检测文件 block 的完整性，HDFS 会记录每个新创建的文件的所有 block 的校验和。当以后检索这些文件的时候，从某个节点获取 block，会首先确认校验和是否一致，如果不一致，会从其他 Datanode 节点上获取该 block 的副本。

3）集群的负载均衡，由于节点的失效或者增加，可能导致数据分布的不均匀，当某个 Datanode 节点的空闲空间大于一个临界值的时候，HDFS 会自动从其他 Datanode 迁移数据过来。

4）Namenode 上的 fsimage 和 edits 日志文件是 HDFS 的核心数据结构，如果这些文件损坏了，HDFS 将失效。因而，Namenode 可以配置成支持维护多 个 FsImage 和 Editlog 的拷贝。任何对 FsImage 或者 Editlog 的修改，都将同步到它们的副本上。它总是选取最近的一致的 FsImage 和 Editlog 使用。Namenode 在 HDFS 是单点存在，如果 Namenode 所在的机器错误，手工的干预是必须的。

5）文件的删除，删除并不是马上从 Namenode 移出 namespace，而是放在 /trash 目录随时可恢复，直到超过设置时间才被正式移除。

再说 Hadoop 本身的容错性，Hadoop 支持升级和回滚，当升级 Hadoop 软件时出现 bug 或者不兼容现象，可以通过回滚恢复到老的 Hadoop 版本。

最后一个就是安全性问题，HDFS 的安全性是比较弱的，只有简单的与 unix 文件系统类似的文件许可控制，未来版本会实现类似 NFS 的 kerberos 验证系统。

总结下：HDFS 作为通用的分布式文件系统并不适合，**它在并发控制、缓存一致性以及小文件读写的效率上是比较弱的。**但是它有自己明确的设计目标，那就是支 持大的数据文件（兆至 T 级），并且这些文件以顺序读为主，以文件读的高吞吐量为目标，并且与 MapReduce 框架紧密结合。

目前作为通用分布式文件系统比较常用的有 **fastDFS**、MogileFS、CEPH 等。可以着重了解一下 fastDFS。