Hadoop核心之——HDFS

- HDFS即Hadoop Distributed File System分布式文件系统，它的设计目标是把超大数据集存储到分布在网络中的多台普通商用计算机上，并且能够提供高可靠性和高吞吐量的服务。分布式文件系统要比普通磁盘文件系统复杂，因为它要引入网络编程，分布式文件系统要容忍节点故障也是一个很大的挑战。

hdfs架构设计

- HDFS主要由3个组件构成，分别是NameNode、SecondaryNameNode和DataNode，HSFS是以master/slave模式运行的，其中NameNode、SecondaryNameNode 运行在master节点，DataNode运行slave节点。

NameNode

- 当一个客户端请求一个文件或者存储一个文件时，它需要先知道具体到哪个DataNode上存取，获得这些信息后，客户端再直接和这个DataNode进行交互，而这些信息的维护者就是NameNode。
- NameNode管理着文件系统命名空间，它维护这文件系统树及树中的所有文件和目录。
  NameNode也负责维护所有这些文件或目录的打开、关闭、移动、重命名等操作。对于实际文件数据的保存与操作，都是由DataNode负责。当一个客户端请求数据时，它仅仅是从NameNode中获取文件的元信息，而具体的数据传输不需要经过NameNode，是由客户端直接与相应的DataNode进行交互。
  NameNode保存元信息的种类有：
- 文件名目录名及它们之间的层级关系
- 文件目录的所有者及其权限
- 每个文件块的名及文件有哪些块组成
  需要注意的是，NameNode元信息并不包含每个块的位置信息，这些信息会在NameNode启动时从各个DataNode获取并保存在内存中，因为这些信息会在系统启动时由数据节点重建。把块位置信息放在内存中，在读取数据时会减少查询时间，增加读取效率。NameNode也会实时通过心跳机制和DataNode进行交互，实时检查文件系统是否运行正常。不过NameNode元信息会保存各个块的名称及文件由哪些块组成。

> 一般来说，一条元信息记录会占用200byte内存空间。假设块大小为64MB，备份数量是3 ，那么一个1GB大小的文件将占用16*3=48个文件块。如果现在有1000个1MB大小的文件，则会占用1000*3=3000个文件块（多个文件不能放到一个块中）。我们可以发现，如果文件越小，存储同等大小文件所需要的元信息就越多，所以，Hadoop更喜欢大文件

元信息的持久化

> 在NameNode中存放元信息的文件是 fsimage。在系统运行期间所有对元信息的操作都保存在内存中并被持久化到另一个文件edits中。并且edits文件和fsimage文件会被SecondaryNameNode周期性的合并（合并过程会在SecondaryNameNode中详细介绍）。

其他问题

为了简化系统的设计，Hadoop只有一个NameNode，这也就导致了hadoop集群的单点故障问题。因此，对NameNode节点的容错尤其重要，hadoop提供了如下两种机制来解决：

- 将hadoop元数据写入到本地文件系统的同时再实时同步到一个远程挂载的网络文件系统（NFS）。
- 运行一个secondary NameNode，它的作用是与NameNode进行交互，定期通过编辑日志文件合并命名空间镜像，当NameNode发生故障时它会通过自己合并的命名空间镜像副本来恢复。需要注意的是secondaryNameNode保存的状态总是滞后于NameNode，所以这种方式难免会导致丢失部分数据（后面会详细介绍）。

DataNode

DataNode是hdfs中的worker节点，它负责存储数据块，也负责为系统客户端提供数据块的读写服务，同时还会根据NameNode的指示来进行创建、删除、和复制等操作。此外，它还会通过心跳定期向NameNode发送所存储文件块列表信息。当对hdfs文件系统进行读写时，NameNode告知客户端每个数据驻留在哪个DataNode，客户端直接与DataNode进行通信，DataNode还会与其它DataNode通信，复制这些块以实现冗余。

SecondaryNameNode

需要注意，SecondaryNameNode并不是NameNode的备份。我们从前面的介绍已经知道，所有HDFS文件的元信息都保存在NameNode的内存中。在NameNode启动时，它首先会加载fsimage到内存中，在系统运行期间，所有对NameNode的操作也都保存在了内存中，同时为了防止数据丢失，这些操作又会不断被持久化到本地edits文件中。

- Edits文件存在的目的是为了提高系统的操作效率，NameNode在更新内存中的元信息之前都会先将操作写入edits文件。在NameNode重启的过程中，edits会和fsimage合并到一起，但是合并的过程会影响到Hadoop重启的速度，SecondaryNameNode就是为了解决这个问题而诞生的。
- SecondaryNameNode的角色就是定期的合并edits和fsimage文件，我们来看一下合并的步骤：
    1. 合并之前告知NameNode把所有的操作写到新的edites文件并将其命名为edits.new。
    2. SecondaryNameNode从NameNode请求fsimage和edits文件
    3. SecondaryNameNode把fsimage和edits文件合并成新的fsimage文件
    4. NameNode从SecondaryNameNode获取合并好的新的fsimage并将旧的替换掉，并把edits用第一步创建的edits.new文件替换掉
    5. 更新fstime文件中的检查点
- fsimage ：保存的是上个检查点的HDFS的元信息
- edits ：保存的是从上个检查点开始发生的HDFS元信息状态改变信息
- fstime：保存了最后一个检查点的时间戳

数据备份

HDFS通过备份数据块的形式来实现容错，除了文件的最后一个数据块外，其它所有数据块大小都是一样的。数据块的大小和备份因子都是可以配置的。NameNode负责各个数据块的备份，DataNode会通过心跳的方式定期的向NameNode发送自己节点上的Block 报告，这个报告中包含了DataNode节点上的所有数据块的列表。
文件副本的分布位置直接影响着HDFS的可靠性和性能。一个大型的HDFS文件系统一般都是需要跨很多机架的，不同机架之间的数据传输需要经过网关，并且，同一个机架中机器之间的带宽要大于不同机架机器之间的带宽。如果把所有的副本都放在不同的机架中，这样既可以防止机架失败导致数据块不可用，又可以在读数据时利用到多个机架的带宽，并且也可以很容易的实现负载均衡。但是，如果是写数据，各个数据块需要同步到不同的机架，会影响到写数据的效率。
而在Hadoop中，如果副本数量是3的情况下，Hadoop默认是这么存放的，把第一个副本放到机架的一个节点上，另一个副本放到同一个机架的另一个节点上，把最后一个节点放到不同的机架上。这种策略减少了跨机架副本的个数提高了写的性能，也能够允许一个机架失败的情况，算是一个很好的权衡。

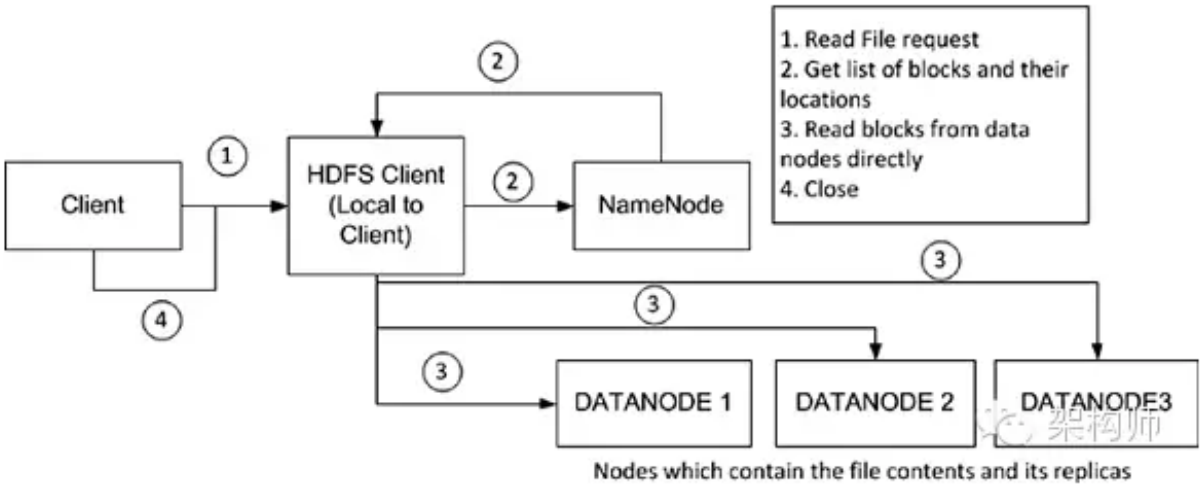
可靠性保证

可以允许DataNode失败。DataNode会定期（默认3秒）的向NameNode发送心跳，若NameNode在指定时间间隔内没有收到心跳，它就认为此节点已经失败。此时，NameNode把失败节点的数据（从另外的副本节点获取）备份到另外一个健康的节点。这保证了集群始终维持指定的副本数。
可以检测到数据块损坏。在读取数据块时，HDFS会对数据块和保存的校验和文件匹配，如果发现不匹配，NameNode同样会重新备份损坏的数据块。

hdfs文件读取过程

hdfs有一个FileSystem实例，客户端通过调用这个实例的open()方法就可以打开系统中希望读取的文件。hdfs通过rpc调用NameNode获取文件块的位置信息，对于文件的每一个块，NameNode会返回含有该块副本的DataNode的节点地址，另外，客户端还会根据网络拓扑来确定它与每一个DataNode的位置信息，从离它最近的那个DataNode获取数据块的副本，最理想的情况是数据块就存储在客户端所在的节点上。
hdfs会返回一个FSDataInputStream对象，FSDataInputStream类转而封装成DFSDataInputStream对象,这个对象管理着与DataNode和NameNode的I/O，具体过程是：

1. 客户端发起读请求
2. 客户端与NameNode得到文件的块及位置信息列表
3. 客户端直接和DataNode交互读取数据
4. 读取完成关闭连接



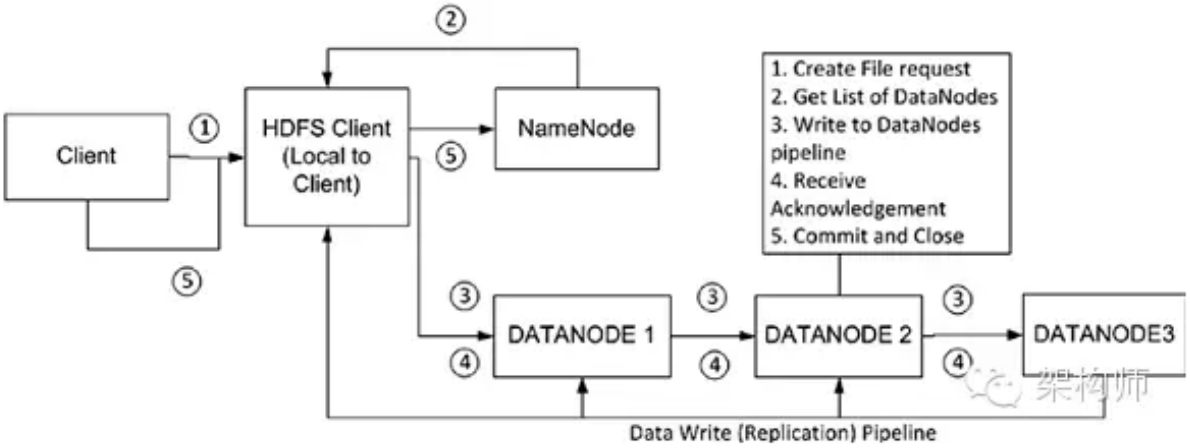
当FSDataInputStream与DataNode通信时遇到错误，它会选取另一个较近的DataNode，并为出故障的DataNode做标记以免重复向其读取数据。FSDataInputStream还会对读取的数据块进行校验和确认，发现块损坏时也会重新读取并通知NameNode。
这样设计的巧妙之处：

1. 让客户端直接联系DataNode检索数据，可以使hdfs扩展到大量的并发客户端，因为数据流就是分散在集群的每个节点上的，在运行MapReduce任务时，每个客户端就是一个DataNode节点。
2. NameNode仅需相应块的位置信息请求（位置信息在内存中，速度极快），否则随着客户端的增加，NameNode会很快成为瓶颈。

hdfs文件写入过程

hdfs有一个DistributedFileSystem实例，客户端通过调用这个实例的create()方法就可以创建文件。DistributedFileSystem会发送给NameNode一个RPC调用，在文件系统的命名空间创建一个新文件，在创建文件前NameNode会做一些检查，如文件是否存在，客户端是否有创建权限等，若检查通过，NameNode会为创建文件写一条记录到本地磁盘的EditLog，若不通过会向客户端抛出IOException。创建成功之后DistributedFileSystem会返回一个FSDataOutputStream对象，客户端由此开始写入数据。
同读文件过程一样，FSDataOutputStream类转而封装成DFSDataOutputStream对象,这个对象管理着与DataNode和NameNode的I/O，具体过程是：

1. 客户端在向NameNode请求之前先写入文件数据到本地文件系统的一个临时文件
2. 待临时文件达到块大小时开始向NameNode请求DataNode信息
3. NameNode在文件系统中创建文件并返回给客户端一个数据块及其对应DataNode的地址列表（列表中包含副本存放的地址）
4. 客户端通过上一步得到的信息把创建临时文件块flush到列表中的第一个DataNode
5. 当文件关闭，NameNode会提交这次文件创建，此时，文件在文件系统中可见



上面第四步描述的flush过程实际处理过程比较负杂，现在单独描述一下：

1. 首先，第一个DataNode是以数据包(数据包一般4KB)的形式从客户端接收数据的，DataNode在把数据包写入到本地磁盘的同时会向第二个DataNode（作为副本节点）传送数据。
2. 在第二个DataNode把接收到的数据包写入本地磁盘时会向第三个DataNode发送数据包
3. 第三个DataNode开始向本地磁盘写入数据包。此时，数据包以流水线的形式被写入和备份到所有DataNode节点
4. 传送管道中的每个DataNode节点在收到数据后都会向前面那个DataNode发送一个ACK,最终，第一个DataNode会向客户端发回一个ACK
5. 当客户端收到数据块的确认之后，数据块被认为已经持久化到所有节点。然后，客户端会向NameNode发送一个确认
6. 如果管道中的任何一个DataNode失败，管道会被关闭。数据将会继续写到剩余的DataNode中。同时NameNode会被告知待备份状态，NameNode会继续备份数据到新的可用的节点
7. 数据块都会通过计算校验和来检测数据的完整性，校验和以隐藏文件的形式被单独存放在hdfs中，供读取时进行完整性校验

hdfs文件删除过程

hdfs文件删除过程一般需要如下几步：

1. 一开始删除文件，NameNode只是重命名被删除的文件到/trash目录，因为重命名操作只是元信息的变动，所以整个过程非常快。在/trash中文件会被保留一定间隔的时间（可配置，默认是6小时），在这期间，文件可以很容易的恢复，恢复只需要将文件从/trash移出即可。
2. 当指定的时间到达，NameNode将会把文件从命名空间中删除
3. 标记删除的文件块释放空间，HDFS文件系统显示空间增加

参数说明

NameNode：是Master节点，是大领导。管理数据块映射；处理客户端的读写请求；配置副本策略；管理HDFS的名称空间；
SecondaryNameNode：是一个小弟，分担大哥namenode的工作量；是NameNode的冷备份；合并fsimage和fsedits然后再发给namenode。
DataNode：Slave节点，奴隶，干活的。负责存储client发来的数据块block；执行数据块的读写操作。
热备份：b是a的热备份，如果a坏掉。那么b马上运行代替a的工作。
冷备份：b是a的冷备份，如果a坏掉。那么b不能马上代替a工作。但是b上存储a的一些信息，减少a坏掉之后的损失。
fsimage:元数据镜像文件（文件系统的目录树。）
edits：元数据的操作日志（针对文件系统做的修改操作记录）
namenode内存中存储的是=fsimage+edits。
SecondaryNameNode负责定时默认1小时，从namenode上，获取fsimage和edits来进行合并，然后再发送给namenode。减少namenode的工作量。