HDFS　原理、架构与特性介绍

本文主要讲述　HDFS 原理 -架构、副本机制、　HDFS 负载均衡、机架感知、健壮性、文件删除恢复机制

1：当前 HDFS 架构详尽分析

HDFS 架构

· NameNode

· DataNode

· Sencondary NameNode

数据存储细节　　NameNode　目录结构

Namenode　的目录结构：

${ dfs.name.dir}/current /VERSION

/edits

/fsimage

/fstime

dfs.name.dir 是 hdfs-site.xml 里配置的若干个目录组成的列表。

NameNode

Namenode 上保存着 HDFS 的名字空间。对于任何对文件系统元数据产生修改的操作, Namenode 都会使用一种称为 EditLog 的事务日志记录下来。 例如，在 HDFS 中创建一个文件, Namenode 就会在 Editlog 中插入一条记录来表示；同样地，修改文件的副本系数也将往 Editlog 插入一条记录。 Namenode 在本地操作系统的文件系统中存储这个 Editlog 。整个文件系统的名 字空间，包括数据块到文件的映射、文件的属性等，都存储在一个称为 FsImage 的文件中，这 个文件也是放在 Namenode 所在的本地文件系统上。

Namenode 在内存中保存着整个文件系统的名字空间和文件数据块映射 (Blockmap) 的映像 。这个关键的元数据结构设计得很紧凑，因而一个有 4G 内存的 Namenode 足够支撑大量的文件 和目录。当 Namenode 启动时，它从硬盘中读取 Editlog 和 FsImage ，将所有 Editlog 中的事务作 用在内存中的 FsImage 上，并将这个新版本的 FsImage 从内存中保存到本地磁盘上，然后删除旧的 Editlog ，因为这个旧的 Editlog 的事务都已经作用在 FsImage 上了。这个过程称为一个检查 点 (checkpoint) 。在当前实现中，检查点只发生在 Namenode 启动时，在不久的将来将实现支持 周期性的检查点。

HDFS NameSpace

HDFS 支持传统的层次型文件组织结构。 用户或者应用程序可以创建目 录，然后将文件保存在这些目录里。文件系统名字空间的层次结构和大多数 现有的文件系统类似：用户可以创建、删除、移动或重命名文件。当前，HDFS 不支持用户磁盘配额和访问权限控制， 也不支持硬链接和软链接。但 是 HDFS 架构并不妨碍实现这些特性。

Namenode 负责维护文件系统命名空间，任何对文件系统名字空间或属 性的修改都将被 Namenode

记录下来。 应用程序可以设置 HDFS 保存的文件 的副本数目。文件副本的数目称为文件的副本系数，这个信息也是由 Namenode 保存的。

DataNode

Datanode 将 HDFS 数据以文件的形式存储在本地的文件系统中， 它并不知道有 关 HDFS 文件的信息。它把每个 HDFS 数据块存储在本地文件系统的一个单独的文件 中。 Datanode 并不在同一个目录创建所有的文件，实际上，它用试探的方法来确定 每个目录的最佳文件数目，并且在适当的时候创建子目录。在同一个目录中创建所 有的本地文件并不是最优的选择，这是因为本地文件系统可能无法高效地在单个目 录中支持大量的文件。

当一个 Datanode 启动时，它会扫描本地文件系统，产生一个这些本地文件对应 的所有 HDFS 数据块的列表，然后作为报告发送到 Namenode ，这个报告就是块状态 报告。

配置 Secondary NameNode

· conf/masters 文件指定的为 Secondary NameNode 节点

· 修改在masters 文件中配置了的机器上的 conf/hdfs-site.xml 文件，加上如下选项：

<property>

<name>dfs.http.address</name>

<value>namenode.hadoop-host.com:50070</value>

</property>

-site.xml：这里有 2 个参数可配置，但一般来说我们不做修改。 fs.checkpoint.period 表示多长时间记录一次 hdfs 的镜像。默认是 1 小时。 fs.checkpoint.size 表示一次记录多大的 size，默认 64M。

<property>

<name>fs.checkpoint.period</name>

<value>3600</value>

<description>The number of seconds between two periodic checkpoints.
</description>

</property>

<property>

<name>fs.checkpoint.size</name>

<value>67108864</value>

<description>The size of the current edit log (in bytes) that triggers a periodic checkpoint even if the fs.checkpoint.period hasn't expired. </description>

</property>

Secondary NameNode

Secondary NameNode 定期合并 fsimage 和 edits 日志，将 edits 日志文件大小控制在一个限度下。

Secondary NameNode 处理流程

(1) 、 namenode 响应 Secondary namenode 请求，将 edit log 推送给 Secondary namenode ， 开始重新写一个新的 edit log 。

(2) 、 Secondary namenode 收到来自 namenode 的 fsimage 文件和 edit log 。

(3) 、 Secondary namenode 将 fsimage 加载到内存，应用 edit log ， 并生成一 个新的 fsimage 文件。

(4) 、 Secondary namenode 将新的 fsimage 推送给 Namenode 。

(5) 、 Namenode 用新的 fsimage 取代旧的 fsimage ， 在 fstime 文件中记下检查 点发生的时

HDFS 通信协议 所有的 HDFS 通讯协议都

是构建在 TCP/IP 协议上。客户端通过一个可 配置的端口连接到 Namenode ， 通过 ClientProtocol 与 Namenode 交互。而 Datanode 是使用 DatanodeProtocol 与 Namenode 交互。再设计上， DataNode 通过周期性的向 NameNode 发送心跳和数据块来保持和 NameNode 的通信，数据块报告的信息包括数据块的属性，即数据块属于哪个文件，数据块 ID ，修改时间等， NameNode 的 DataNode 和数据块的映射 关系就是通过系统启动时 DataNode 的数据块报告建立的。从 ClientProtocol 和 Datanodeprotocol 抽象出一个远程调用 （RPC ），在设计上， Namenode 不会主动发起 RPC ， 而是是响应来自客户端和 Datanode 的 RPC 请求。

HDFS 的安全模式 Namenode 启动后会进入一个称为安全模式的特殊状态。处于安全模式 的 Namenode 是不会进行数据块的复制的。 Namenode 从所有的 Datanode 接收心跳信号和块状态报告。 块状态报告包括了某个 Datanode 所有的数据 块列表。每个数据块都有一个指定的最小副本数。 当 Namenode 检测确认某 个数据块的副本数目达到这个最小值，那么该数据块就会被认为是副本安全 (safely replicated) 的；在一定百分比（这个参数可配置）的数据块被 Namenode 检测确认是安全之后（加

上一个额外的 30 秒等待时间），Namenode 将退出安全模式状态。接下来它会确定还有哪些数据块的副本没有达到指定数目，并将这些数据块复制到其他 Datanode 上。 2：HDFS 文件读取的解析

文件读取流程

流程分析

·使用HDFS 提供的客户端开发库 Client，向远程的 Namenode 发起 RPC 请求；

· Namenode 会视情况返回文件的部分或者全部 block 列表，对于每个 block，Namenode 都会返回有该 block 拷贝的 DataNode 地址；

·客户端开发库 Client 会选取离客户端最接近的 DataNode 来读取 block ；如果客户端本身就是 DataNode，那么将从本地直接获取数据 .

·读取完当前 block 的数据后，关闭与当前的 DataNode 连接，并为读取下一个 block 寻找最佳的 DataNode ；

· 当读完列表的 block 后，且文件读取还没有结束，客户端开发库会继续向 Namenode 获取下一批的 block 列表。

· 读取完一个 block 都会进行 checksum 验证，如果读取 datanode 时出现错误，客户端会通知 Namenode ，然后再从下一个拥有该 block 拷贝的 datanode 继续读。 3 :HDFS 文件写入的解析

文件写入流程

流程分析

· 使用HDFS 提供的客户端开发库 Client ，向远程的 Namenode 发起 RPC 请求；

· Namenode会检查要创建的文件是否已经存在， 创建者是否有权限进行操作，成功则会为文件 创建一个记录，否则会让客户端抛出异常；

· 当客户端开始写入文件的时候，会将文件切分成多个 packets ，并在内部以数据队列 "data queue" 的形式管理这些 packets ，并向 Namenode 申请新的 blocks ，获取用来存储 replicas 的合适的 datanodes 列表，列表的大小根据在

Namenode 中对 replication 的设置而定。

· 开始以pipeline（管道）的形式将 packet 写入所有的 replicas 中。把 packet 以流的方式写入第一个 datanode ，该 datanode 把该 packet 存储之后，再将其传递给在此 pipeline 中的下一个 datanode ，直到最后一个 datanode ，这种写数据的方式呈流水线的形式。

· 最后一个datanode 成功存储之后会返回一个 ack packet ，在 pipeline 里传递至客户端，在客户端的开发库内部维护着 "ack queue" ，成功收到 datanode 返回的 ack packet 后会从 "ack queue" 移除相应的 packet 。

· 如果传输过程中，有某个 datanode 出现了故障，那么当前的 pipeline 会被关闭，出现故障的 datanode 会从当前的 pipeline 中移除，剩余的 block 会继续剩下的 datanode 中继续以 pipeline 的形式传输，同时 Namenode 会分配一个新的 datanode ，保持 replicas 设定的数量。

流水线复制

当客户端向 HDFS 文件写入数据的时候，一开始是写到本地临时文件中。假设该文件的副 本系数设

置为 3 ，当本地临时文件累积到一个数据块的大小时，客户端会从 Namenode 获取一个 Datanode 列表用于存放副本。然后客户端开始向第一个 Datanode 传输数据，第一个 Datanode 一小部分一小部分 (4 KB) 地接收数据，将每一部分写入本地仓库，并同时传输该部分到列表中 第二个 Datanode 节点。第二个 Datanode 也是这样，一小部分一小部分地接收数据，写入本地 仓库，并同时传给第三个 Datanode 。最后，第三个 Datanode 接收数据并存储在本地。因此， Datanode 能流水线式地从前一个节点接收数据，并在同时转发给下一个节点，数据以流水线的 方式从前一个 Datanode 复制到下一个

更细节的原理

客户端创建文件的请求其实并没有立即发送给 Namenode ，事实上，在刚开始阶 段 HDFS 客户端会先将文件数据缓存到本地的一个临时文件。应用程序的写操作被透 明地重定向到这个临时文件。当这个临时文件累积的数据量超过一个数据块的大小 ，客户端才会联系 Namenode 。 Namenode 将文件名插入文件系统的层次结构中，并 且分配一个数据块给它。然后返回 Datanode 的标识符和目标数据块给客户端。接着 客户端将这块数据从本地临时文件上传到指定的 Datanode 上。当文件关闭时，

在临　时文件中剩余的没有上传的数据也会传输到指定的 Datanode　上。然后客户端告诉　Namenode　文件已经关闭。此时　Namenode　才将文件创建操作提交到日志里进行存储　。如果　Namenode　在文件关闭前宕机了，　则该文件将丢失。　4：副本机制

特点

1. 数据类型单一

2. 副本数比较多

3. 写文件时副本的放置方法

4. 动态的副本创建策略

5. 弱化的副本一致性要求

副本摆放策略

修改副本数

1. 集群只有三个　Datanode ， hadoop　系统 replication=4　时，

会出现什么情况？

对于上传文件到 hdfs 上时，当时 hadoop 的副本系数是几，这个文件的块数副本数就会有几份，无论以后你怎么更改系统副本系统，这个文件的副本数都不会改变，也就说上传到分布式系统上的文件副本数由当时的系统副本数决定，不会受 replication 的更改而变化，除非用命令来更改文件的副本数。因为 dfs.replication 实质上是 client 参数，在 create 文件时可以指定具体 replication ，属性 dfs.replication 是不指定具体 replication 时的采用默认备份数。文件上传后，备份数已定，修改 dfs.replication 是不会影响以前的文件的，也不会影响后面指定备份数的文件。只影响后面采用默认备份数的文件。但可以利用 hadoop 提供的命令后期改某文件的备份数： hadoop fs -setrep -R 1 。如果你是在 hdfs-site.xml 设置了 dfs.replication ，这并一定就得了，因为你可能没把 conf 文件夹加入到你的 project 的 classpath 里，你的程序运行时取的 dfs.replication 可能是 hdfs-default.xml 里的 dfs.replication ，默认是 3。可能这个就是造成你为什么 dfs.replication 老是 3 的原因。你可以试试在创建文件时， 显式设定 replication 。replication 一般到 3 就可以了，大了意义也不大。 5：HDFS 负载均衡

HDFS 的数据也许并不是非常均匀的分布在各个

DataNode 中。一个常见的原因是在现有的集群上经常会增添新的 DataNode 节点。当新增一个数据块（一个文件的数据被保存在一系列的块中） 时, NameNode 在选择 DataNode 接收这个数据块之前，会考虑到很多因素。其中的一些考虑的是：

· 将数据块的一个副本放在正在写这个数据块的节点上。

· 尽量将数据块的不同副本分布在不同的机架上， 这样集群可在完全失去某一机架的情况下还能存活。

· 一个副本通常被放置在和写文件的节点同一机架的某个节点上，这样可以减少跨越机架的网络 I/O。

· 尽量均匀地将 HDFS 数据分布在集群的 DataNode 中。 6：

HDFS 机架感知

HDFS 机架感知

通常，大型 Hadoop 集群是以机架的形式来组织的，同一个机架上不同 节点间的网络状况比不同机架之间的更为理想。 另外， NameNode 设法将 数据块副本保存在不同的机架上以提高容错性。

而 HDFS 不能够自动判断集群中各个 datanode 的网络拓扑情况 Hadoop 允 许集群的管理员通过配置 dfs.network.script 参数来确定节点所处的机架。 文 件提供了 IP->rackid 的翻译。 NameNode 通过这个得到集群中各个 datanode 机器的 rackid 。 如果 topology.script.file.name 没有设定，则每个 IP 都会翻译成 / default-rack 。

有了机架感知， NameNode 就可以画出上图所示的 datanode 网络拓扑图。 D1,R1 都是交换机，最底层是 datanode 。 则 H1 的 rackid=/D1/R1/H1 ， H1 的 parent 是 R1 ， R1 的是 D1 。 这些 rackid 信息可以通过 topology.script.file.name 配置。有了这些 rackid 信息就可以计算出任意两台 datanode 之间的距离。

distance(/D1/R1/H1,/D1/R1/H1)=0 相同的 datanode

distance(/D1/R1/H1,/D1/R1/H2)=2　　同一　rack　下的不同 datanode

distance(/D1/R1/H1,/D1/R1/H4)=4　　同一　IDC　下的不同 datanode

distance(/D1/R1/H1,/D2/R3/H7)=6　　不同　IDC　下的 datanode　　7：HDFS 访问

访问方式

HDFS　给应用提供了多种访问方式。　用户可以通过 Java API　接口访问，也　可以通过　C 语言的封装　API 访问，还可以通过浏览器的方式访问　　HDFS　中的文件。

8：HDFS　健壮性

HDFS　的主要目标就是即使在出错的情况下也要保证数据存储的可靠性。　常见的三种出错情况是：Namenode　出错　, Datanode　出错和网络割裂　(network partitions)　。

磁盘数据错误，心跳检测和重新复制

每个　Datanode　节点周期性地向　Namenode 发送心跳信号。网络割裂可能　导致一部分　Datanode　跟 Namenode　失去联系。　Namenode　通过心跳信号的缺　失

来检测这一情况，并将这些近期不再发送心跳信号

Datanode 标记为宕机 ，不会再将新的 IO 请求发给它们。

任何存储在宕机 Datanode 上的数据将不 再有效。

Datanode 的宕机可能会引起一些数据块的副本系数低于指

定值， Namenode 不断地检测这些需要复制的数据块，一

旦发现就启动复制操作。 在下列情况下，可能需要重新复

制：某个 Datanode 节点失效，某个副本遭 到损坏，

Datanode 上的硬盘错误，或者文件的副本系数增大。

数据完整性

从某个 Datanode 获取的数据块有可能是损

坏的，损坏可能是由 Datanode 的存储设备错误、 网络错误

或者软件 bug 造成的。 HDFS 客户端软 件实现了对

HDFS 文件内容的校验和 (checksum) 检查。当客户端创建

一个新 的 HDFS 文件，会计算这个文件每个数据块的校验

和，并将校验和作为一个 单独的隐藏文件保存在同一个

HDFS 名字空间下。当客户端获取文件内容后 ，它会检验

从 Datanode 获取的数据跟相应的校验和文件中的校验和

是否匹 配，如果不匹配， 客户端可以选择从其他 Datanode

获取该数据块的副本。

元数据磁盘错误

FsImage 和 Editlog 是 HDFS 的核心数据结构。如果这些文件损坏了，整个 HDFS 实例都将失效。因而， Namenode 可以配置成支持维护多个 FsImage 和 Editlog 的副本。任何对 FsImage 或者 Editlog 的修改，都将同步到它们的副 本上。这种多副本的同步操作可能会降低 Namenode 每秒处理的名字空间事 务数量。然而这个代价是可以接受的， 因为即使 HDFS 的应用是数据密集的 ，它们也非元数据密集的。当 Namenode 重启的时候，它会选取最近的完整 的 FsImage 和 Editlog 来使用。

Namenode 是 HDFS 集群中的单点故障(single point of failure) 所在。如果 Namenode 机器故障，是需要手工干预的。目前，自动重启或在另一台机器 上做 Namenode 故障转移的功能还没实现。

快照

快照支持某一特定时刻的数据的复制备份。利用快照， 可以让 HDFS 在 数据损坏时恢复到过去一个已知正确的时间点。 HDFS 目前还不支持快照功 能，但计划在将来的版本进行支持。 9：HDFS 文件删除恢复机制

当用户或应用程序删除某个文件时，这个文件并没有立刻从 HDFS 中删 除。实际上， HDFS 会将这

个文件重命名转移到 /trash 目录。只要文件还在 /trash 目录中，该文件就可以被迅速地恢复。文件在 /trash 中保存的时间是可 配置的，当超过这个时间时， Namenode 就会将该文件从名字空间中删除。 删除文件会使得该文件相关的数据块被释放。注意，从用户删除文件到 HDFS 空闲空间的增加之间会有一定时间的延迟。

只要被删除的文件还在 /trash 目录中，用户就可以恢复这个文件。 如果 用户想恢复被删除的文件， 他 / 她可以浏览 /trash 目录找回该文件。 /trash 目 录仅仅保存被删除文件的最后副本。 /trash 目录与其他的目录没有什么区别 ，除了一点： 在该目录上 HDFS 会应用一个特殊策略来自动删除文件。目前 的默认策略是删除 /trash 中保留时间超过 6 小时的文件。将来，这个策略可以 通过一个被良好定义的接口配置。 开启回收站

hdfs-site.xml

<configuration>

<property>

<name>fs.trash.interval</name>

<value> 1440 </value>

<description>Number ofminutes between trash checkpoints.

If zero, the trashfeature is disabled.

</description>

</property>

</configuration>

1, fs.trash.interval 参数设置保留时间为 1440 分钟 (1 天)

2，回收站的位置：在 HDFS 上的 /user/$USER/.Trash/Current/

10：HDFS 分布式缓存（DistributedCache ）

（1）在 HDFS 上准备好要共享的数据 (text、 archive 、 jar) ,

你拼路径的时候必须加前缀 "file://" 说明是本地路径，否则 hadoop 默认访问的路径是 hdfs。

（2）DistributedCache 在 Mapper 或者 Reducer 启动时会被 copy to local ，然后被 DistributedCache.getLocalCacheFiles() 调用，运行完 job 后 local cache file 会被删掉，如果另一个 job 也需要这样一份文件，需要重新添加、重新缓存，因为在分布式场景下 task 并不知道该 node 是否存在 cache file 。如果在同台机器已经有了 dist cache file，不会再次 download ，DistributedCache 根据缓存文档修改的时间戳进行追踪。在作业执行期间，当前应用程序或者外部程序不能修改缓存文件，所以分布式缓存一般用来缓存只读文件。

（3）DistributedCache 在添加的时候注意要添加具体的文件，如果你添加目录，DistributedCache 将不会自动遍历、识别目录下的文件。 11：HDFS 缺点

大量小文件

因为 Namenode 把文件系统的元数据放置在内存中，所以文件系统所能 容纳的文件数目是由 Namenode 的内存大小来决定。 一般来说， 每一个文件 、文件夹和 Block 需要占据 150 字节左右的空间， 所以，如果你有 100 万个文件，每一个占据一个 Block ，你就至少需要 300MB 内存。当前来说， 数百万 的文件还是可行的， 当扩展到数十亿时，

对于当前的硬件水平来说就没法实　　现了。还有一个问题就是，因为　Map task　的数量是由　splits　来决定的，所以　用 MR　处理大量的小文件时，就会产生过多的　　Maptask　，线程管理开销将会　增加作业时间。举个例子，处理　　10000M 的文件，若每个　split　为　1M　，那就会　有　10000　个 Maptasks　，会有很大的线程开销；　若每个　split　为　100M　，则只有　100　个　Maptasks　，每个　Maptask　将会有更多的事情做，而线程的管理开销也　　将减小很多。