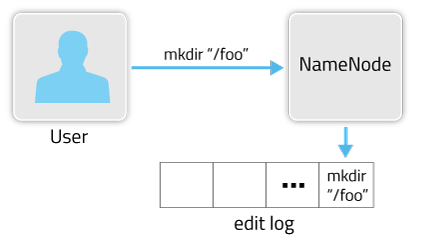
Hadoop Checkpoint

在Hadoop分布式文件系统，Checkpoint是将数据修改（元数据）持久化的机制，是NameNode恢复及重启的重要部分。在HDFS中，对于文件数据的修改不是直接写到磁盘，很多操作是缓存到内存的Buffer中，当遇到一个检查点Checkpoint时，系统会强制将内存中的数据写回磁盘，此时才会记录日志，从而产生持久的修改状态。

Namenode的主要功能是维护HDFS的Namespace，包括目录树，文件权限和blocksID与文件的映射关系等。元数据的修改必须持久化到稳定的磁盘空间中，而且需要做到容错。

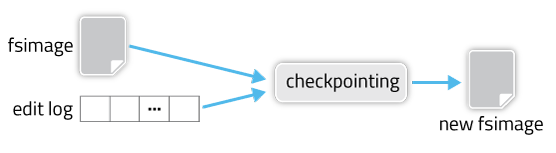
文件系统的元数据包括两种文件，fsimage和edit log，fsimage是文件系统元数据的某个时间检查点的数据，包括文件系统的所有目录和文件inode的序列化信息的一个快照，fsimage不适合做文件的增量更新（例如重命名某个文件）。因此，为了解决这个问题，Namenode使用edit log记录文件的所有操作。当Namenode崩溃时，首先使用加载fsimage恢复Namenode的执行，然后使用edit log 重放文件的所有变化（尽量恢复文件系统的最近状态），edit log由一些列文件组成（称为edit log segments），这些文件构成了fsimage构建以来文件系统的所有变化。

下图是Namenode持久化文件系统元数据的流程图:



edits在每次修改HDFS时都会插入记录，但是fsimage则在整个HDFS运行期间不会发生变化，在hadoop初始版本中，仅在namenode启动时合并fsimage和edits文件，然后把edits清空。

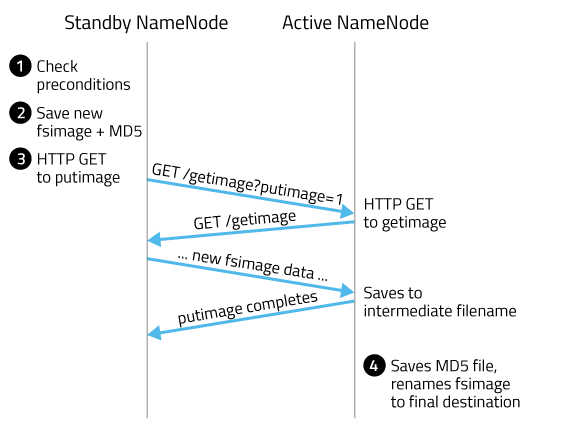
但是edits文件可能会增长到很大，下次启动Namenode时需要花费很长时间来恢复，另一方面，如果在hdfs运行过程中发生Namenode故障，那么edits中的记录就会丢失，所以需要使用checkpoint将修改操作持久化，FSimage和edits的工作机制如下图：



Checkpoint的触发，由下面两个参数：dfs.namenode.checkpoint.check.period和dfs.namenode.checkpoint.size，分别是时间和日志记录的大小，来配置，默认是1个小时或者64MB。

# 1.Standby NameNode(SNN)

NameNode配置HA后，Active和Standby NameNode有一个共享的Edits存储目录。一般有三个或者更多的JournalNodes维护。流程如下图所示：

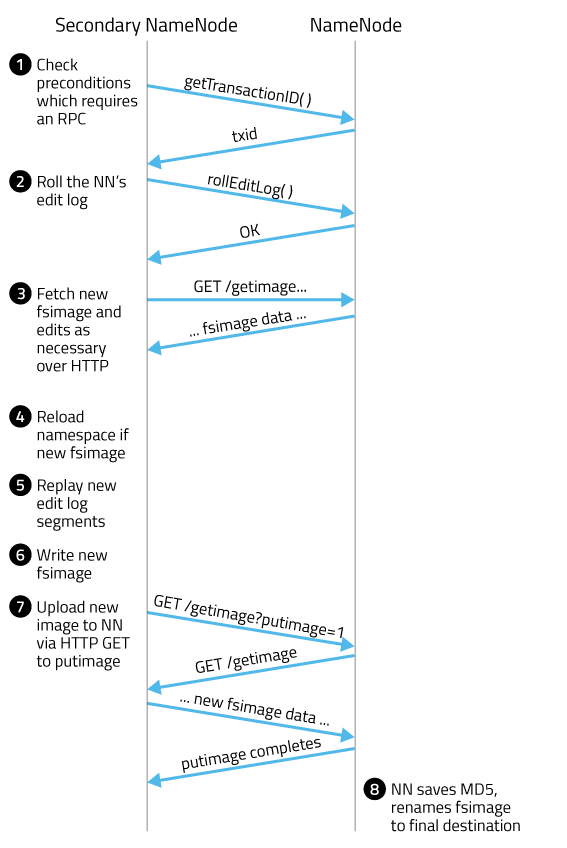


Active Namenode向共享的edits目录中写入数据，Standby Namenode从目录中周期的获取相对较新的namespace。因此，checkpointing在满足上面的两个条件时，将namespace写入新的fsimage，类似于saveNamespace命令，然后通过HTTP将新的fsimage文件上传到Active Namenode（ANN）的目录中。流程如下所示：

1. SbNN检查时间或者edit文件是否满足条件，耗费的时间从上次checkpoint开始，edits逐渐堆积
2. SbNN将namespace保存到新的fsimage文件中，命名为fsimage.ckpt\_，txid是最近edit log事务的编号。SbNN同时为fsimage生成md5文件后，重命名为fsimage\_\*。当进行fsimage生成时，预示着NameNode failover或者SbNN Web UI。但是ANN中的常规的HDFS客户端操作，例如list,读写文件等不会受到影响
3. SbNN发送HTTP Get命令到NN的GetImageServlet（/getimage?putimage=1）中，这个URL参数中有fsimage的transaction ID及SbNN 主机名和http 端口号
4. ANN servlet使用Get请求中的信息，反过来向SbNN发送GetImageServlet请求。类似于Standby，同样将新的fsimage命令为临时文件名fsimage.ckpt\_t，并创建MD5文件后，命名为fsimage\_\*

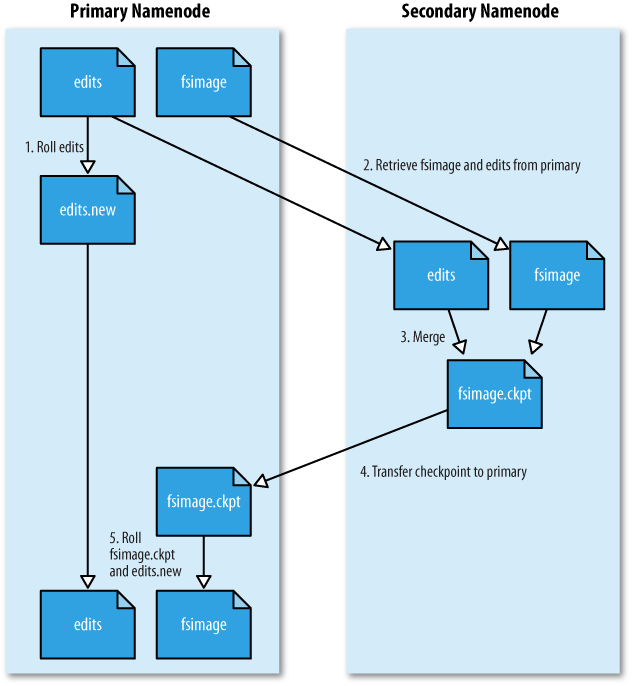
# 2.Secondary NameNode(2NN)

在非HA部署下，Checkpointing是由SecondaryNameNode来完成，由于没有共享的目录，SNN在进行checkpoint操作时，首先更新Namespace，过程如下图所示：



过程如下所示：

1. 2NN检查时间或者edit文件是否满足条件，最近的edit log事务ID使用NameNodeProtocol
2. #getTransactionId的RPC调用从NameNode中获取
3. 2NN触发edit log roll操作，停止当前edit log segment的操作并且启动新的edit log。NN向新的edit log写入操作。2NN对先前的edit进行操作，与fsimage进行合并，并返回当前fsimage和rolled的edit的transaction ID。在HA的配置下，不需要edit log的roll操作，因为standby NameNode定期的对edit log进行rolls操作
4. 在整个流程中，会出现两个transactin ID，2NN根据需要使用Get命令从NNs GetImageServlet中获取新的fsimage及edits文件。2NN中会有前一次checkpoint的一些文件
5. 在必要的情况下，2NN从最新下载的fsimage中加载namespace
6. 5)2NN为了获取最新的transaction ID，replay新的edit log，以下的步骤就和StandbyNamenode相同
7. 6)2NN将namespace写入新的fsimage中
8. 7)2NN通过HTTP GET(/getimage?putimage=1)与NN进行交互，NN从2NN中下载最新的fsimage文件



# 3.Checkpoint Node及Backup Node

Checkpoint Node和2NN的功能类似，是为了解决Namenode在运行过程中仅将文件操作写入edit log的问题（edit log太大，下次启动时会耗费大量时间）。Checkpoint Node周期的从Namenode中获取fsimage和edits文件，然后将两个文件合并，合并后将fsimage上传到NameNode中，但是2NN不提供上传fsimage的功能。

Backup Node，功能也类似，但是其与NameNode同步，维护edits文件流，不需要周期的从Namenode获取fsimage和edit文件。Backup Node将当前状态保存在内存中，当创建checkpoint时保存image文件。

# 4.Checkpoint的导入和恢复

当Namenode发生故障丢失数据时，可以使用2NN进行fsimage的导入恢复，过程如下：

1. 在Namenode节点上创建dfs.name.dir指定的目录
2. 在hdfs-site.xml中指定配置文件dfs.namenode.checkpoint.dir
3. 启动Namenode时带上选项-importCheckpoint

Namenode首先会将dfs.namenode.checkpoint.dir中的文件拷贝到dfs.name.dir中，如果此时dfs.name.dir中已经包含了合法的fsimage文件（Namenode没有发生元数据丢失却执行了导入操作），那么Namenode就会执行失败。否则，Namenode会检测导入的fsimage文件是否与文件系统中的数据一致，若一致则成功完成导入恢复。另外一点，checkpoing对于edit文件很大时启动Namenode较长，因为需要花费较长的时间将edits日志merge到fsimage中。

Namenode还提供了一种Recovery的方式，在配置文件中hdfs-site.xml中配置两个参数：

1) dfs.namenode.name.dir，指定namenode中元数据的本地存储位置，这里位置可以指定多个，并且用逗号分开

2) dfs.namenode.name.dir.restore，布尔变量，若为true，在每次checkpoint的时候会自动恢复丢失数据的dir

可见，Recovery只是在本地存储了多个元数据的备份，虽然可以避免文件损害带来的故障，但是对于节点故障来说，本地多个备份都不能使用，有一定的局限性，不过一般情况下都会把多个本地备份设置在不同的物理磁盘分区上，当一个分区坏掉以后可以使用另一个。

为了使Recovery生效，需要使NameNode在Recovery Mode下启动，启动的命令方式如下：namenode -recover。在Recovery Mode下，Namenode使用交互式的方式提示用户恢复数据。如果不想提示，直接恢复，则使用-force选项。但是，由于recovery mode可能会使你丢失数据，所有在使用时需要备份edit log和fsimage文件。

# 5.Checkpoint的影响

当Namenode的文件操作使Edit log达到数百GB的数据，将磁盘占满，可能会造成NN的崩溃。当这种情况发生时，只有重启NN，然后replay所有的edits，这个过程将会很长。

Checkpoint是I/O及网络密集操作的过程，将会影响client的性能。在大规模集群，处理数百万文件及多GB fsimage时，将fsimage文件写回Namenode可能会消耗大量的可用带宽。这种情况下，可以使用dfs.image.transfer.bandwidthPerSec来限制传送速度。配置这个参数时，同样要根据预估的时间调整dfs.image.transfer.timeout，延长发送时间。

参考文献：

http://blog.cloudera.com/blog/2014/03/a-guide-to-checkpointing-in-hadoop/

http://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-hdfs/HdfsUserGuide.html#Recovery\_Mode

http://morrisjobke.de/2013/12/11/Hadoop-NameNode-and-siblings/