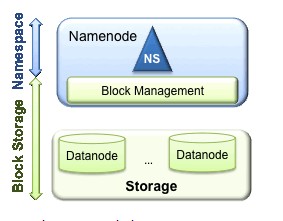
HDFS Federation源码分析

hdfs主要包含两层结构：

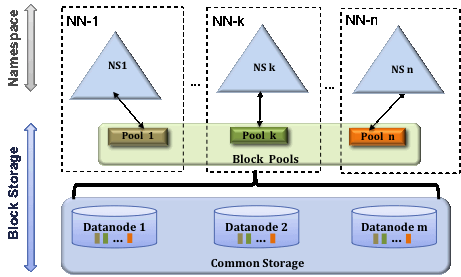


1. NameSpace，管理目录、文件和数据块，支持常见的文件系统操作，如创建、修改、删除文件等
2. Block Storage Service，包含两个部分：

* Block Management，在NameNode中运行，通过心跳维护集群中dn的基本关系；根据心跳中的block reports维护blocks的位置；支持数据块基本操作（创建、删除及管理副本的复制和存放）
* Physical Storage，存储实际的数据块并提供对数据块的读写服务。这部分分别在DN上实现。

在上述的架构中，整个集群中存在一个namespace，而该namespace被仅有的一个Active NN管理。这个架构的最大的问题是HDFS的扩展性，其底层存储（DN）是水平扩展，当集群存储空间不够时，可简单的添加机器进行水平扩展，但namespace只存放在单个NN上，内存中存储这个分布式文件系统的元数据信息，单台集群的服务能力是固定的，这限制了集群中数据块、文件和目录的数目，而且单NN的吞吐量比较低。

为了解决这个问题，HDFS中开发了Federation模式，其原理如下图所示：



Federation中，使用多个NN来使HDFS的Namespace能够水平扩展，NN之间相互独立并且不需要相互协调。每个DN被所有的NN用作公共存储块的地方，每个DN都会向所在集群中所有的NN注册，并且会周期性的发送心跳和块信息报告，同时处理来自NN的指令。

# 1.概念

1. Block Pool

属于单个Namespace的一组Block。DN为每一个Block Pool存储块，可以将DN看做是一个物理概念，而Block Pool是一个重新将Block划分的逻辑概念，同一个DN可以存着属于多个block pool的多个块。

Block Pool由属于同一个Namespace的数据块组成，内部自治，不会与其它Block Pool交互。因此一个NameNode挂掉，不会影响其他NN。

1. NameSpace Volume

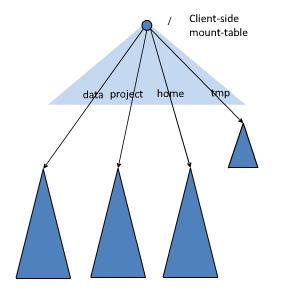
NN的namespace和对应的block pool一起被称为namespace volume，它是管理的基本单位，当一个NN被删除，其所有的datanode上对应的block pool也要相应删除。

1. ClusterID

在HDFS Federation中使用ClusterID来区分集群中的每个节点，当格式化NN时，这个ClusterID会自动生成，也可以手动指定。

1. Multi Namespaces

在集群中多个命名空间，相互之间会进行数据的共享和访问。使用全局唯一的命名空间是解决数据共享和访问的一种方法。可以使用Client Side Mount Table做到数据的共享和访问：



深色三角型代表一个独立的命名空间，浅色代表从客户端角度去访问下方的子命名空间。各个深色的命名空间Mount到浅色的表中，客户可以根据不同的挂载点来访问不同的命名空间，这如同在Linux系统中访问不同的挂载点是一样。推荐使用ViewFs来模拟Unix/Linux系统中的Client side mount table。

# 2.DN与NN的交互

DataNode与NameNode交互主要集中在DataNode中，类结构如下图所示：



为了支持Federation,以逻辑层面划分Block-pool和block-pool下的Block分布，并以block-pool为单位来与NameNode进行相关的通信。在DataNode中，BPServiceActor实例和Active及Standby模式的NameNode实例进行交互，是任务的执行者，一个BPServiceActor代表一个BlockPool（对应一个独立的NameSpace），其中BPOfferService管理和每个NaneNode进行管理实际的BPServiceActor实例，其类成员变量如下：

*class BPOfferService {*

*//本Block-pool服务代表的NameSpace信息，和Namenode握手的第一阶段分配所得*

*NamespaceInfo bpNSInfo;*

*//所在DataNode相关的注册信息，和NameNode握手的第二阶段分配所得*

*volatile DatanodeRegistration bpRegistration;*

*//所属Datanode实例*

*private final DataNode dn;*

*private final String nameserviceId;*

*private volatile String bpId;*

*//代表和当前Active状态的Namenode关联的BPServiceActor实例，如果所有NN处于standby，此属性可以为空*

*private BPServiceActor bpServiceToActive = null;*

*//本NameService服务下指向所有Namenode的BPServiceActor实例，不管是active或者standby*

*private final List<BPServiceActor> bpServices =*

*new CopyOnWriteArrayList<BPServiceActor>();*

*//根据NN的地址来初始化BPServiceActor，并加入到bpService集合中*

*BPOfferService(*

*final String nameserviceId,*

*List<InetSocketAddress> nnAddrs,*

*List<InetSocketAddress> lifelineNnAddrs,*

*DataNode dn) {*

*this.nameserviceId = nameserviceId;*

*this.dn = dn;*

*for (int i = 0; i < nnAddrs.size(); ++i) {*

*this.bpServices.add(new BPServiceActor(nnAddrs.get(i),*

*lifelineNnAddrs.get(i), this));*

*}*

*}*

其主要下面四个功能：

* 预先与NameNode进行握手，获取NN的namespace，源码如下所示：

*NamespaceInfo retrieveNamespaceInfo() throws IOException {*

*NamespaceInfo nsInfo = null;*

*while (shouldRun()) {*

*try {*

*nsInfo = bpNamenode.versionRequest(); ......*

*break;*

*} .......*

*return nsInfo;*

*}*

* 向NameNode注册storage，从获取返回的registrationID

*void register(NamespaceInfo nsInfo) throws IOException {*

*DatanodeRegistration newBpRegistration = bpos.createRegistration();*

*while (shouldRun()) {*

*try {*

*// Use returned registration from namenode with updated fields*

*newBpRegistration = bpNamenode.registerDatanode(newBpRegistration);*

*newBpRegistration.setNamespaceInfo(nsInfo);*

*bpRegistration = newBpRegistration;*

*break;*

*} ......*

*}*

*bpos.registrationSucceeded(this, bpRegistration);*

*// random short delay - helps scatter the BR from all DNs*

*scheduler.scheduleBlockReport(dnConf.initialBlockReportDelayMs);*

*}*

* 周期性向NameNode发送心跳

*while (shouldRun()) {*

*try {*

*offerService();*

*}…..*

*}*

在offerService中，与NN进行心跳，DN根据NN发送的心跳信息获取命令

*private void offerService() throws Exception {*

*while (shouldRun()) {*

*try {*

*final boolean sendHeartbeat = scheduler.isHeartbeatDue(startTime);*

*HeartbeatResponse resp = null;*

*if (sendHeartbeat) {*

*boolean requestBlockReportLease = (fullBlockReportLeaseId == 0) &&*

*scheduler.isBlockReportDue(startTime);*

*if (!dn.areHeartbeatsDisabledForTests()) {*

*//向NameNode发送心跳*

*resp = sendHeartBeat(requestBlockReportLease);*

*.....*

*bpos.updateActorStatesFromHeartbeat(this, resp.getNameNodeHaState());*

*state = resp.getNameNodeHaState().getState();*

*if (state == HAServiceState.ACTIVE) {*

*handleRollingUpgradeStatus(resp);*

*}*

*}*

*if (ibrManager.sendImmediately() || sendHeartbeat) {*

*ibrManager.sendIBRs(bpNamenode, bpRegistration,*

*bpos.getBlockPoolId(), dn.getMetrics());*

*}*

*List<DatanodeCommand> cmds = null;*

*boolean forceFullBr =*

*scheduler.forceFullBlockReport.getAndSet(false);*

*if (forceFullBr) {*

*LOG.info("Forcing a full block report to " + nnAddr);*

*}*

*if ((fullBlockReportLeaseId != 0) || forceFullBr) {*

*//根据NameNode发送过来的心跳信息，获取命令*

*cmds = blockReport(fullBlockReportLeaseId);*

*fullBlockReportLeaseId = 0;*

*}*

*processCommand(cmds == null ? null : cmds.toArray(new DatanodeCommand[cmds.size()]));*

*if (!dn.areCacheReportsDisabledForTests()) {*

*DatanodeCommand cmd = cacheReport();*

*processCommand(new DatanodeCommand[]{ cmd });*

*}*

*}*

*}*

在sendHeartbeat中，汇报dn的存储信息，包括StorageReport，capacity及已使用的存储等信息。

* 处理NameNode发送过来的命令

BlockPoolManager类主要用于管理DN的BPOfferService，对BPOfferService对象的创建、删除、启动、停止及关闭的操作都需要通过BlockPoolManager提供的方法控制，代码如下：

*class BlockPoolManager {*

*private final Map<String, BPOfferService> bpByNameserviceId =*

*Maps.newHashMap();*

*private final Map<String, BPOfferService> bpByBlockPoolId =*

*Maps.newHashMap();*

*private final List<BPOfferService> offerServices =*

*new CopyOnWriteArrayList<>();*

*private final DataNode dn;*

*private final Object refreshNamenodesLock = new Object();*

*BlockPoolManager(DataNode dn) {*

*this.dn = dn;*

*}*

*}*

DN启动过程中与NN交互操作，都是在DN类中进行，包括握手、注册、数据块上报和发送心跳等。

# 3.Block Scanner

VolumeScanner类用于扫描block-pool下的block文件并校验文件是否损坏，对block和最后的校验时间进行跟踪。一个DN对应一个BlockScanner，BlockScanner对不同的Storage ID的blockScanner进行管理。

启动BlockPoolSliceStorage用于管理DN上对应同一个block pool的BlockPoolSlices集合，由于一个DN可能挂载多个存储设备，即逻辑上对应多个Volume，一个BlockPoolSlice对应一个Volume，所以在同一个DN上同一个Block Pool，可以管理多个BlockPoolSlice。BlockPoolSliceStorage的主要职能如下：

* 对新生成的block-pool对应的存储进行格式化

*private void format(StorageDirectory bpSdir, NamespaceInfo nsInfo) throws IOException {*

*bpSdir.clearDirectory(); // create directory*

*this.layoutVersion = HdfsServerConstants.DATANODE\_LAYOUT\_VERSION;*

*this.cTime = nsInfo.getCTime();*

*this.namespaceID = nsInfo.getNamespaceID();*

*this.blockpoolID = nsInfo.getBlockPoolID();*

*writeProperties(bpSdir);*

*}*

* 恢复存储状态以保持一致性
* 在升级的时候对block-pool进行快照处理
* 回滚block-pool到上一个快照
* 删除快照并提交block

# 3.Block Report

DataNode使用BPOfferService的blockReport向NN汇报block信息：BPOfferService#

scheduleBlockReport：

*void scheduleBlockReport(long delay) {*

*for (BPServiceActor actor : bpServices) {*

*actor.getScheduler().scheduleBlockReport(delay);*

*}*

*}*

其中调用blockReport方法，源码所示：

*//获取所属的blockPoolID，对应的block List*

*Map<DatanodeStorage, BlockListAsLongs> perVolumeBlockLists =*

*dn.getFSDataset().getBlockReports(bpos.getBlockPoolId());*

*//初始化reports列表*

*StorageBlockReport reports[] =*

*new StorageBlockReport[perVolumeBlockLists.size()];*

*for(Map.Entry<DatanodeStorage, BlockListAsLongs> kvPair : perVolumeBlockLists.entrySet()) {*

*BlockListAsLongs blockList = kvPair.getValue();*

*reports[i++] = new StorageBlockReport(kvPair.getKey(), blockList);*

*totalBlockCount += blockList.getNumberOfBlocks();*

*}*

*try {*

*if (totalBlockCount < dnConf.blockReportSplitThreshold) {*

*// Below split threshold, send all reports in a single message.*

*DatanodeCommand cmd = bpNamenode.blockReport(*

*bpRegistration, bpos.getBlockPoolId(), reports,*

*new BlockReportContext(1, 0, reportId, fullBrLeaseId));*

*}......*

NameNode获取到DN的Report后，进行blockReport：

*public DatanodeCommand blockReport(final DatanodeRegistration nodeReg,*

*String poolId, final StorageBlockReport[] reports,*

*final BlockReportContext context) throws IOException {*

*//block的管理由BlockManager的管理*

*final BlockManager bm = namesystem.getBlockManager();*

*boolean noStaleStorages = false;*

*for (int r = 0; r < reports.length; r++) {*

*final BlockListAsLongs blocks = reports[r].getBlocks();*

*final int index = r;*

*noStaleStorages = bm.runBlockOp(new Callable<Boolean>() {*

*@Override*

*public Boolean call() throws IOException {*

*//根据blocks信息，进行处理*

*return bm.processReport(nodeReg, reports[index].getStorage(),*

*blocks, context);*

*}*

*});*

*}*

*......*

*}*

在federation模式下，为了区分blockPoolID，只将对应的blockPoolId的block信息发送该对应的nameNode，主要的源码如下：

*Map<DatanodeStorage, BlockListAsLongs> perVolumeBlockLists =*

*dn.getFSDataset().getBlockReports(bpos.getBlockPoolId());*

FsDatasetSpi根据blockPoolId获取FsDatasetImpl#blockReports：

*@Override*

*public Map<DatanodeStorage, BlockListAsLongs> getBlockReports(String bpid) {*

*.......*

*for (ReplicaInfo b : volumeMap.replicas(bpid)) {*

*String volStorageID = b.getVolume().getStorageID();*

*continue;*

*}*

*}*

*}*

*.......*

*for (FsVolumeImpl v : curVolumes) {*

*blockReportsMap.put(v.toDatanodeStorage(),*

*builders.get(v.getStorageID()).build());*

*}*

*return blockReportsMap;*

*}*

volumeMap:ReplicaMap，

*private final Map<String, LightWeightResizableGSet<Block, ReplicaInfo>> map =*

*new HashMap<String, LightWeightResizableGSet<Block, ReplicaInfo>>();*

ReplicaMap.replicas(bpId)，基于bpID获取BlockID

http://itfish.net/article/11750.html