尚硅谷大数据技术之Hadoop HA

# 第1章 Hadoop HA高可用

## 1.1 HA概述

**1）**所谓HA（High Availablity），即高可用（7\*24小时不中断服务）。

**2）**实现高可用最关键的策略是消除单点故障。HA严格来说应该分成各个组件的HA机制：HDFS的HA和YARN的HA。

**3）** Hadoop2.0之前，在HDFS集群中NameNode存在单点故障SPOF（Single Points Of Failure）。

**4）**NameNode主要在以下两个方面影响HDFS集群

NameNode机器发生意外，如宕机，集群将无法使用，直到管理员重启

NameNode机器需要升级，包括软件、硬件升级，此时集群也将无法使用

HDFS HA功能通过配置多台Active / Standby角色的NameNode实现在集群中对NameNode的热备来解决上述问题。如果出现故障，如机器崩溃或机器需要升级维护，这时可通过此种方式将NameNode很快的切换到另外一台机器。

## 1.2 HDFS-HA工作机制

通过多NameNode消除单点故障

## 1.3 HDFS-HA手动故障转移

### 1.3.1 工作要点

**1）**元数据管理方式需要改变

内存中各自保存一份元数据；

Edits日志只有Active状态的NameNode节点可以做写操作；

两个NameNode都可以读取Edits；

共享的Edits放在一个共享存储中管理（qjournal和NFS两个主流实现）

**2）**必须保证多NameNode之间能够ssh无密码登录

**3）**隔离（Fence），即同一时刻仅仅有一个NameNode对外提供服务

### 1.3.2 配置HDFS-HA集群

**1）在opt目录下创建一个ha文件夹**

[atguigu@hadoop202 opt]$ mkdir ha

**2）将/opt/module/下的 hadoop-3.1.3拷贝到/opt/ha目录下**

[atguigu@hadoop202 opt] cp -r hadoop-3.1.3 ha

**3）配置core-site.xml**

<configuration>

<!-- 指定NameNode的地址 -->

<property>

<name>fs.defaultFS</name>

<value>hdfs://mycluster</value>

</property>

<!-- 指定hadoop数据的存储目录

官方配置文件中的配置项是hadoop.tmp.dir ,用来指定hadoop数据的存储目录,此次配置用的hadoop.data.dir是自己定义的变量， 因为在hdfs-site.xml中会使用此配置的值来具体指定namenode 和 datanode存储数据的目录

-->

<property>

<name>hadoop.data.dir</name>

<value>/opt/module/ha/hadoop-3.1.3/data</value>

</property>

</configuration>

**4）配置hdfs-site.xml**

<configuration>

<!-- 指定副本数 -->

<property>

<name>dfs.replication</name>

<value>3</value>

</property>

<property>

<name>dfs.nameservices</name>

<value>mycluster</value>

</property>

<!-- 指定NameNode数据的存储目录 -->

<property>

<name>dfs.namenode.name.dir</name>

<value>file://${hadoop.data.dir}/name</value>

</property>

<!-- 指定Datanode数据的存储目录 -->

<property>

<name>dfs.datanode.data.dir</name>

<value>file://${hadoop.data.dir}/data</value>

</property>

<!-- 声明journalnode服务器存储目录-->

<property>

<name>dfs.journalnode.edits.dir</name>

<value>/opt/module/ha/hadoop-3.1.3/data/jn</value>

</property>

<!-- 集群中NameNode节点都有哪些 -->

<property>

<name>dfs.ha.namenodes.mycluster</name>

<value>nn1,nn2,nn3</value>

</property>

<!-- nn1的RPC通信地址 -->

<property>

<name>dfs.namenode.rpc-address.mycluster.nn1</name>

<value>hadoop102:9000</value>

</property>

<!-- nn2的RPC通信地址 -->

<property>

<name>dfs.namenode.rpc-address.mycluster.nn2</name>

<value>hadoop103:9000</value>

</property>

<!-- nn3的RPC通信地址 -->

<property>

<name>dfs.namenode.rpc-address.mycluster.nn3</name>

<value>hadoop104:9000</value>

</property>

<!-- nn1的http通信地址 -->

<property>

<name>dfs.namenode.http-address.mycluster.nn1</name>

<value>hadoop102:9870</value>

</property>

<!-- nn2的http通信地址 -->

<property>

<name>dfs.namenode.http-address.mycluster.nn2</name>

<value>hadoop103:9870</value>

</property>

<!-- nn3的http通信地址 -->

<property>

<name>dfs.namenode.http-address.mycluster.nn3</name>

<value>hadoop104:9870</value>

</property>

<!-- 指定NameNode元数据在JournalNode上的存放位置 -->

<property>

<name>dfs.namenode.shared.edits.dir</name>

<value>qjournal://hadoop102:8485;hadoop103:8485;hadoop104:8485/mycluster</value>

</property>

<!-- 配置隔离机制，即同一时刻只能有一台服务器对外响应 -->

<property>

<name>dfs.ha.fencing.methods</name>

<value>sshfence</value>

</property>

<!-- 使用隔离机制时需要ssh无秘钥登录-->

<property>

<name>dfs.ha.fencing.ssh.private-key-files</name>

<value>/home/atguigu/.ssh/id\_rsa</value>

</property>

<!-- 访问代理类：client用于确定哪个NameNode为Active -->

<property> <name>dfs.client.failover.proxy.provider.mycluster</name>

<value>org.apache.hadoop.hdfs.server.namenode.ha.ConfiguredFailoverProxyProvider</value>

</property>

<!-- 兼容配置，先跳过 -->

<property>

<name>dfs.client.datanode-restart.timeout</name>

<value>30s</value>

</property>

</configuration>

**5）拷贝配置好的hadoop环境到其他节点**

### 1.3.3 启动HDFS-HA集群

**0）**修改HADOOP的环境变量 ，指向HA的集群

sudo vim /etc/profile.d/my\_env.sh

HADOOP\_HOME=/opt/module/ha/hadoop-3.1.3

**2）**各个JournalNode节点上，输入以下命令启动journalnode服务

hdfs --daemon start journalnode

**3）**在[nn1]上，对其进行格式化，并启动

hdfs namenode -format

hdfs --daemon start namenode

**4）**在[nn2]和[nn3]上，同步nn1的元数据信息

hdfs namenode -bootstrapStandby

**5）**启动[nn2] 和2 [nn3]

hdfs --daemon start namenode

**6）**查看web页面显示，102 103 104 都为standby状态

**7）**启动所有datanode

hdfs --daemon start datanode

**8）**将[nn1]切换为Active

hdfs haadmin -transitionToActive nn1

**9）**是否Active

hdfs haadmin -getServiceState nn1

**10）**kill掉Active的NameNode，进行手动故障转移

## 1.4 配置HDFS-HA自动故障转移

### 1.4.1 工作要点

前面学习了使用命令hdfs haadmin手动进行故障转移，在该模式下，即使现役NameNode已经失效，系统也不会自动从现役NameNode转移到待机NameNode，下面学习如何配置部署HA自动进行故障转移。自动故障转移为HDFS部署增加了两个新组件：ZooKeeper和ZKFailoverController（ZKFC）进程。ZooKeeper是维护少量协调数据，通知客户端这些数据的改变和监视客户端故障的高可用服务。

**1）HA的自动故障转移依赖于ZooKeeper的以下功能**：

（1）故障检测：

集群中的每个NameNode在ZooKeeper中维护了一个持久会话，如果机器崩溃，ZooKeeper中的会话将终止，ZooKeeper通知另一个NameNode需要触发故障转移。

（2）现役NameNode选择：

ZooKeeper提供了一个简单的机制用于唯一的选择一个节点为active状态。如果目前现役NameNode崩溃，另一个节点可能从ZooKeeper获得特殊的排外锁以表明它应该成为现役NameNode。

**2）ZKFC**

ZKFC是自动故障转移中的另一个新组件，是ZooKeeper的客户端，也监视和管理NameNode的状态。每个运行NameNode的主机也运行了一个ZKFC进程，ZKFC负责：

（1）**健康监测：**ZKFC使用一个健康检查命令定期地ping与之在相同主机的NameNode，只要该NameNode及时地回复健康状态，ZKFC认为该节点是健康的。如果该节点崩溃，冻结或进入不健康状态，健康监测器标识该节点为非健康的。

（2）**ZooKeeper会话管理：**当本地NameNode是健康的，ZKFC保持一个在ZooKeeper中打开的会话。如果本地NameNode处于active状态，ZKFC也保持一个特殊的znode锁，该锁使用了ZooKeeper对短暂节点的支持，如果会话终止，锁节点将自动删除。

（3）**基于ZooKeeper的选择：**如果本地NameNode是健康的，且ZKFC发现没有其它的节点当前持有znode锁，它将为自己获取该锁。如果成功，则它已经赢得了选择，并负责运行故障转移进程以使它的本地NameNode为Active。故障转移进程与前面描述的手动故障转移相似，首先如果必要保护之前的现役NameNode，然后本地NameNode转换为Active状态。



图1-1 HDFS-HA故障转移机制

### 1.4.2 规划集群

表1-1

|  |  |  |
| --- | --- | --- |
| hadoop102 | hadoop103 | hadoop104 |
| NameNode | NameNode | NameNode |
| ZKFC | ZKFC | ZKFC |
| JournalNode | JournalNode | JournalNode |
| DataNode | DataNode | DataNode |
| ZK | ZK | ZK |
|  | ResourceManager |  |
| NodeManager | NodeManager | NodeManager |

### 1.4.3 配置Zookeeper集群

**1） 集群规划**

在hadoop102、hadoop103和hadoop104三个节点上部署Zookeeper。

**2）** **解压安装**

（1）解压Zookeeper安装包到/opt/module/目录下

[atguigu@hadoop102 software]$ tar -zxvf zookeeper-3.5.7.tar.gz -C /opt/module/

（2）在/opt/module/zookeeper-3.5.7/这个目录下创建zkData

[atguigu@hadoop102 zookeeper-3.5.7]$ mkdir -p zkData

（3）重命名/opt/module/zookeeper-3.5.7/conf这个目录下的zoo\_sample.cfg为zoo.cfg

[atguigu@hadoop102 conf]$ mv zoo\_sample.cfg zoo.cfg

**3）** **配置zoo.cfg文件**

（1）具体配置

[atguigu@hadoop102 conf]$ vim zoo.cfg

dataDir=/opt/module/zookeeper-3.5.7/zkData

增加如下配置

#######################cluster##########################

server.2=hadoop102:2888:3888

server.3=hadoop103:2888:3888

server.4=hadoop104:2888:3888

（2）配置参数解读

Server.A=B:C:D。

A是一个数字，表示这个是第几号服务器；

B是这个服务器的IP地址；

C是这个服务器与集群中的Leader服务器交换信息的端口；

D是万一集群中的Leader服务器挂了，需要一个端口来重新进行选举，选出一个新的Leader，而这个端口就是用来执行选举时服务器相互通信的端口。

集群模式下配置一个文件myid，这个文件在dataDir目录下，这个文件里面有一个数据就是A的值，Zookeeper启动时读取此文件，拿到里面的数据与zoo.cfg里面的配置信息比较从而判断到底是哪个server。

**4）集群操作**

（1）在/opt/module/zookeeper-3.5.7/zkData目录下创建一个myid的文件

[atguigu@hadoop102 zkData] touch myid

添加myid文件，注意一定要在linux里面创建，在notepad++里面很可能乱码

（2）编辑myid文件

[atguigu@hadoop102 zkData] vim myid

在文件中添加与server对应的编号：如2

（3）拷贝配置好的zookeeper到hadoop103 hadoop104

[atguigu@hadoop102 module] xsync zookeeper-3.5.7

并分别修改myid文件中内容为3、4

（4）分别启动zookeeper

[atguigu @hadoop102 zookeeper-3.5.7]# bin/zkServer.sh start

[atguigu @hadoop103 zookeeper-3.5.7]# bin/zkServer.sh start

[atguigu @hadoop104 zookeeper-3.5.7]# bin/zkServer.sh start

（5）查看状态

[atguigu @hadoop102 zookeeper-3.5.7]# bin/zkServer.sh status

JMX enabled by default

Using config: /opt/module/zookeeper-3.4.10/bin/../conf/zoo.cfg

Mode: follower

[atguigu @hadoop103 zookeeper-3.5.7]# bin/zkServer.sh status

JMX enabled by default

Using config: /opt/module/zookeeper-3.4.10/bin/../conf/zoo.cfg

Mode: leader

[atguigu @hadoop104 zookeeper-3.5.7]# bin/zkServer.sh status

JMX enabled by default

Using config: /opt/module/zookeeper-3.4.10/bin/../conf/zoo.cfg

Mode: follower

### 1.4.4 配置HDFS-HA自动故障转移

**1）具体配置**

（1）在hdfs-site.xml中增加

<property>

<name>dfs.ha.automatic-failover.enabled</name>

<value>true</value>

</property>

（2）在core-site.xml文件中增加

<property>

<name>ha.zookeeper.quorum</name>

<value>hadoop102:2181,hadoop103:2181,hadoop104:2181</value>

</property>

**2） 启动**

（1）关闭所有HDFS服务：

[atguigu @hadoop102 hadoop-3.1.3]# stop-dfs.sh

（2）启动Zookeeper集群，在每个Zookeeper节点执行

[atguigu @hadoop102 zookeeper-3.5.7]# bin/zkServer.sh start

（3）初始化HA在Zookeeper中状态：

[atguigu @hadoop102 hadoop-3.1.3]# hdfs zkfc -formatZK

（4）启动HDFS服务：

[atguigu @hadoop102 hadoop-3.1.3]# start-dfs.sh

**3）web端查看，一个Active的NameNode， 两个Standby的NameNode**

**4）**验证

（1）将Active NameNode进程kill,实现自动故障转移

[atguigu @hadoop102 hadoop-3.1.3]# kill -9 namenode的进程id

## 1.5 YARN-HA配置

### 1.5.1 YARN-HA工作机制

**1）** **YARN-HA工作机制，如图1-2所示**

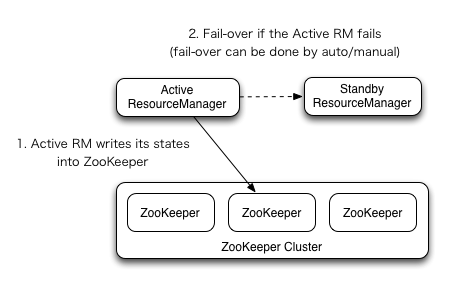


图1-2 YARN-HA工作机制

### 1.5.2 配置YARN-HA集群

**1）**规划集群

表1-2

|  |  |  |
| --- | --- | --- |
| hadoop102 | hadoop103 | hadoop104 |
| NameNode | NameNode | NameNode |
| JournalNode | JournalNode | JournalNode |
| ZKFC | ZKFC | ZKFC |
| DataNode | DataNode | DataNode |
| ZK | ZK | ZK |
| ResourceManager | ResourceManager | ResourceManager |
| NodeManager | NodeManager | NodeManager |

**2）** 具体配置

（1）yarn-site.xml

<configuration>

<property>

<name>yarn.nodemanager.aux-services</name>

<value>mapreduce\_shuffle</value>

</property>

<!--启用resourcemanager ha-->

<property>

<name>yarn.resourcemanager.ha.enabled</name>

<value>true</value>

</property>

<!--声明HA resourcemanager的地址-->

<property>

<name>yarn.resourcemanager.cluster-id</name>

<value>cluster-yarn1</value>

</property>

<!-- 指定RM的逻辑列表 -->

<property>

<name>yarn.resourcemanager.ha.rm-ids</name>

<value>rm1,rm2,rm3</value>

</property>

<!-- =========== rm1 配置============ -->

<!-- 指定rm1 的主机名 -->

<property>

<name>yarn.resourcemanager.hostname.rm1</name>

<value>hadoop102</value>

</property>

<!-- 指定rm1的web端地址 -->

<property>

<name>yarn.resourcemanager.webapp.address.rm1</name>

<value>hadoop102:8088</value>

</property>

<!-- 指定rm1的内部通信地址 -->

<property>

<name>yarn.resourcemanager.address.rm1</name>

<value>hadoop102:8032</value>

</property>

<!-- 指定AM向rm1申请资源的地址 -->

<property>

<name>yarn.resourcemanager.scheduler.address.rm1</name>

<value>hadoop102:8030</value>

</property>

<!-- 指定供NM连接的地址 -->

<property>

<name>yarn.resourcemanager.resource-tracker.address.rm1</name>

<value>hadoop102:8031</value>

</property>

<!-- =========== rm2 配置============ -->

<property>

<name>yarn.resourcemanager.hostname.rm2</name>

<value>hadoop103</value>

</property>

<property>

<name>yarn.resourcemanager.webapp.address.rm2</name>

<value>hadoop103:8088</value>

</property>

<property>

<name>yarn.resourcemanager.address.rm2</name>

<value>hadoop103:8032</value>

</property>

<property>

<name>yarn.resourcemanager.scheduler.address.rm2</name>

<value>hadoop103:8030</value>

</property>

<property>

<name>yarn.resourcemanager.resource-tracker.address.rm2</name>

<value>hadoop103:8031</value>

</property>

<!-- =========== rm3 配置============ -->

<property>

<name>yarn.resourcemanager.hostname.rm3</name>

<value>hadoop104</value>

</property>

<property>

<name>yarn.resourcemanager.webapp.address.rm3</name>

<value>hadoop104:8088</value>

</property>

<property>

<name>yarn.resourcemanager.address.rm3</name>

<value>hadoop104:8032</value>

</property>

<property>

<name>yarn.resourcemanager.scheduler.address.rm3</name>

<value>hadoop104:8030</value>

</property>

<property>

<name>yarn.resourcemanager.resource-tracker.address.rm3</name>

<value>hadoop104:8031</value>

</property>

<!--指定zookeeper集群的地址-->

<property>

<name>yarn.resourcemanager.zk-address</name>

<value>hadoop102:2181,hadoop103:2181,hadoop104:2181</value>

</property>

<!--启用自动恢复-->

<property>

<name>yarn.resourcemanager.recovery.enabled</name>

<value>true</value>

</property>

<!--指定resourcemanager的状态信息存储在zookeeper集群-->

<property>

<name>yarn.resourcemanager.store.class</name> <value>org.apache.hadoop.yarn.server.resourcemanager.recovery.ZKRMStateStore</value>

</property>

<!-- 环境变量的继承 -->

<property>

<name>yarn.nodemanager.env-whitelist</name>

<value>JAVA\_HOME,HADOOP\_COMMON\_HOME,HADOOP\_HDFS\_HOME,HADOOP\_CONF\_DIR,CLASSPATH\_PREPEND\_DISTCACHE,HADOOP\_YARN\_HOME,HADOOP\_MAPRED\_HOME</value>

</property>

</configuration>

（2）同步配置文件到其他节点

[atguigu@hadoop202 hadoop]$ xsync yarn-site.xml

**3）启动HDFS**

**4）启动YARN**

（1）在hadoop102中执行：

[atguigu@hadoop202 hadoop]$ start-yarn.sh

（2）查看服务状态

[atguigu@hadoop202 hadoop]$ yarn rmadmin -getServiceState rm1

（3）web端查看YARN的状态

## 1.6 HDFS Federation架构设计

**1）**NameNode架构的局限性

（1）Namespace（命名空间）的限制

由于NameNode在内存中存储所有的元数据（metadata），因此单个NameNode所能存储的对象（文件+块）数目受到NameNode所在JVM的heap size的限制。50G的heap能够存储20亿（200million）个对象，这20亿个对象支持4000个DataNode，12PB的存储（假设文件平均大小为40MB）。随着数据的飞速增长，存储的需求也随之增长。单个DataNode从4T增长到36T，集群的尺寸增长到8000个DataNode。存储的需求从12PB增长到大于100PB。

（2）隔离问题

由于HDFS仅有一个NameNode，无法隔离各个程序，因此HDFS上的一个实验程序就很有可能影响整个HDFS上运行的程序。

（3）性能的瓶颈

由于是单个NameNode的HDFS架构，因此整个HDFS文件系统的吞吐量受限于单个NameNode的吞吐量。

**2）HDFS Federation架构设计**

表1-3

|  |  |  |
| --- | --- | --- |
| NameNode | NameNode | NameNode |
| 元数据 | 元数据 | 元数据 |
| Log | machine | 电商数据/话单数据 |



图1-3 HDFS Federation架构设计

**3）HDFS Federation应用思考**

不同应用可以使用不同NameNode进行数据管理,如图片业务、爬虫业务、日志审计业务等。

Hadoop生态系统中，不同的框架使用不同的NameNode进行管理NameSpace（隔离性）。