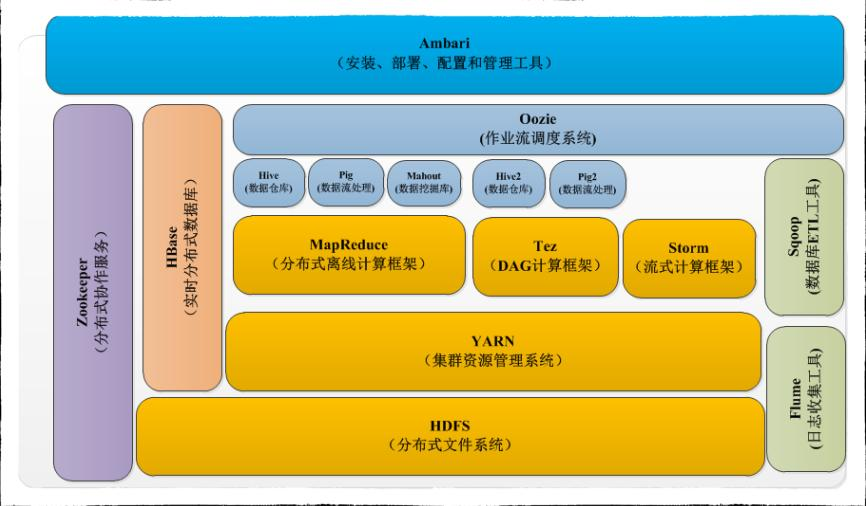
hadoop 2.7.6

MapReduce 分布式计算框架

YARN 集群资源管理系统

HDFS 分布式文件系统



• HDFS(Hadoop分布式文件系统)

• Mapreduce(分布式计算框架) （开发）

• Zookeeper(分布式协作服务)

• Hbase(分布式列存数据库) （开发）

• Hive(基于Hadoop的数据仓库) （开发）

• Sqoop(数据同步工具)

• Pig(基于Hadoop的数据流系统) （开发）

• Mahout(数据挖掘算法库) （开发）

• Flume(日志收集工具)

HDFS 角色及概念

– Client

– Namenode

– Secondarynode

– Datanode

• NameNode 不能做热备份，数据可能不是实时，真正有变更时需要等Secondarynode 定期推送获取最新信息

– Master节点,管理HDFS的名称空间和数据块映射信息,配置副本策略,处理所有客户端请求。

• Secondary NameNode

– 定期合并 fsimage（数据块的映射信息） 和fsedits（数据变更日志）,推送给NameNode

– 紧急情况下,可辅助恢复NameNode

• DataNode

– 数据存储节点,存储实际的数据

– 汇报存储信息给NameNode。

• Client

– 切分文件

– 访问HDFS

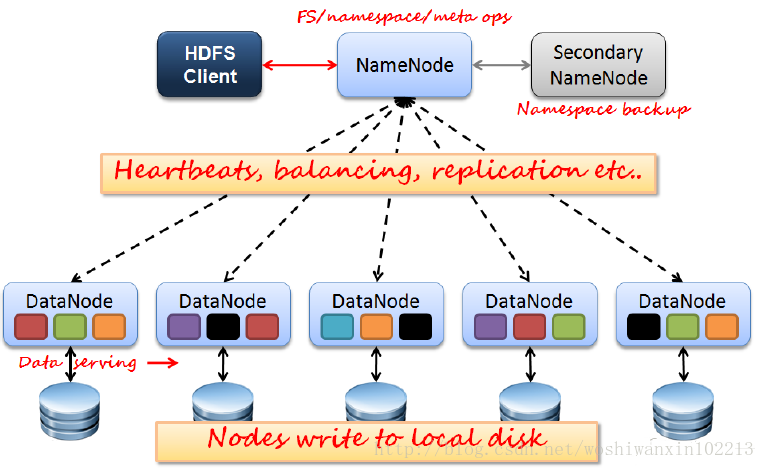
– 不NameNode交互,获取文件位置信息

– 不DataNode交互,读取和写入数据。

• Block

– 每块缺省64MB / 128MB大小 数据切块

– 每块可以多个副本（备份）存在不同机器上



1.HDFS Client 将数据进行切分，然后访问NameNode--->

2.NameNode（管理者角色）处理所有客户端请求，获取存储文件的位置,记录fsimage（数据块的映射信息）--->

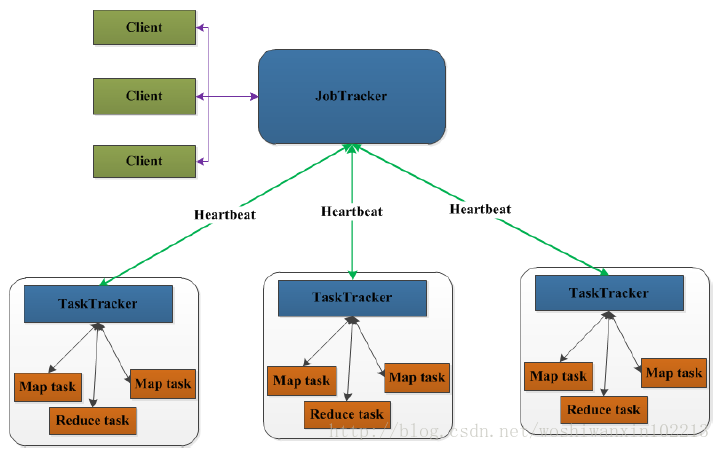
3.将数据存储到真正的DataNode中，存储完数据要回送消息给NameNode

Secondarynode 定期合并 fsimage（数据块的映射信息） 和fsedits（数据变更日志）,推送给NameNode

HDFS的空间约等于所有DataNode主机的空间之和

每个DataNode的空间最好一样大

MapReduce



• 角色和概念

– JobTracker

– TaskTracker

– Map Task （开发）

– Reducer Task （开发）

• JobTracker

– Master节点,只有一个

– 管理所有作业

– 作业/任务的监控、错误处理等

– 将任务分解成一系列任务,并分派给TaskTracker

• TaskTracker

– Slave节点,一般是多台

– 运行Map Task和Reduce Task

– 并与JobTracker交互,汇报任务状态。

• Map Task:解析每条数据记录,传递给用户编写的map(),幵执行,将输出结果写入本地磁盘(如果为map-only作业,直接写入HDFS)。

• Reducer Task:从Map Task的执行结果中,远程读取输入数据,对数据进行排序,将数据按照分组传递给用户编写的reduce凼数执行。

JobTracker（只有一个Master，管理节点）负责接收Client请求，分配给底层的TaskTracker（Slave）进行运算，TaskTracker计算完将计算结果返回给JobTracker，再返回给Client客户端

Yarn

• Yarn 角色

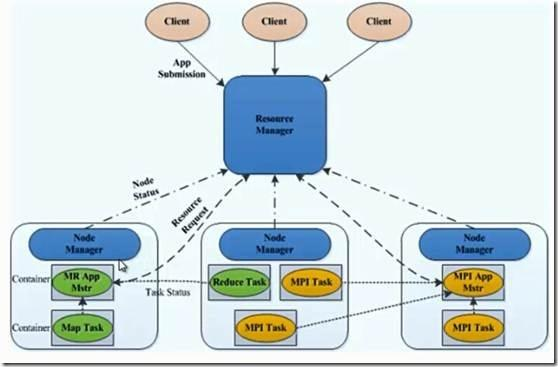
– Resourcemanager

– Nodemanager

– ApplicationMaster

– Container

– Client



• ResourceManager

– 处理客户端请求

– 启动 / 监控 ApplicationMaster

– 监控 NodeManager

– 资源分配不调度

• NodeManager 节点管理代理

– 单个节点上的资源管理

– 处理来自 ResourceManager 的命令

– 处理来自 ApplicationMaster 的命令

• Container 计算CPU 、内存等资源使用

– 对任务运行行环境的抽象,封装了 CPU 、内存等

– 多维资源以及环境变量、启动命令等任务运行相关的信息资源分配不调度

• ApplicationMaster ResourceManager启动的一个程序

– 数据切分

– 为应用程序申请资源,并分配给内部任务

– 任务监控不容错

• Client

– 用户不 YARN 交互的客户端程序

– 提交应用程序、监控应用程序状态,杀死应用程序等

• YARN 的核心思想

• 将 JobTracker 和 TaskTacker 进行分离,它由下面几大构成组件:

– ResourceManager 一个全局的资源管理器

– NodeManager 每个节点(RM)代理

– ApplicationMaster 表示每个应用

– 每一个 ApplicationMaster 有多个 Container 在NodeManager 上运行

Hadoop 安装配置

Hadoop 单机模式安装配置

1./etc/hostname node1

2./etc/hosts 192.168.1.11 node1

3.装包

yum -y install java-1.8.0-openjdk-devel

(安装 Openjdk 和 openjdk-devel) jps

hadoop-2.7.6.tar.gz

tar -xf hadoop-2.7.6.tar.gz

mv hadoop-2.7.6 /usr/local/hadoop

设置JAVA\_HOME环境变量

rpm -ql java-1.8.0-openjdk

vim /usr/local/hadoop/etc/hadoop/hadoop-env.sh

JAVA\_HOME="/usr/lib/jvm/java-1.8.0-openjdk-1.8.0.131-11.b12.el7.x86\_64/jre"

HADOOP\_CONF\_DIR="/usr/local/hadoop/etc/hadoop"

– 测试

cd /usr/local/hadoop/bin

./hadoop version

./bin/hadoop

统计热词

jar ./share/hadoop/mapreduce/hadoop-mapreduce-examples-2.7.3.jar --jar路径 wordcount input output

回车有说明

./bin/hadoop -----hadoop解释器

jar

./share/hadoop/mapreduce/hadoop-mapreduce-examples-2.7.3.jar jar包路径

wordcount 方法

input 要统计的目录（可以多个目录）

output 统计完的结果放入的目录（输出目录要不能存在）

Hadoop 伪分布式 （可以做测试环境）

• 伪分布式

– 伪分布式的安装和完全分布式类似,但区别是所有角色安装在一台机器上,使用本地磁盘,一般生产环境都会使用完全分布式,伪分布式一般用来学习和测试方面的功能

– 伪分布式的配置和完全分布式配置类似

完全分布式

HDFS 分布式文件系统

• 系统规划

主机 角色 软件

192.168.1.10 namenode NameNode HDFS

SecondaryNameNode

ResourceManager YARN

192.168.1.11 Node1 DataNode HDFS

NodeManager YARN

192.168.1.12 Node2 DataNode HDFS

NodeManager YARN

192.168.1.13 node3 DataNode HDFS

NodeManager YARN

hostname 最好不要有特殊符号

namenode 存储映射信息 磁盘

datanode 存储真正数据 磁盘

resourcemanager 分配计算 cpu 内存 读取数据切分

nodemanager 计算 cpu 内存 坏了，整个集群数据全部丢失

1. 所有虚拟机都要配置/etc/hosts，四台之间要互相通信
2. 所有虚拟机装java-1.8.0

3.• 配置 SSH 信任关系(在namenode配置)

– 注意:不能出现要求输入 yes 的情况,namenode连每台机器都要能登录成功,包括本机!!!

vim /etc/ssh/ssh\_config

Host \*

GSSAPIAuthentication yes

StrictHostKeyChecking no

– ssh-keygen -b 2048 -t rsa -N '' -f key（文件名）

– ssh-copy-id -i ./key.pub [root@ip.xx.xx.xx](mailto:root@ip.xx.xx.xx)

4.• HDFS 完全分布式系统配置

– 环境配置文件 hadoop-env.sh

JAVA\_HOME="/usr/lib/jvm/java-1.8.0-openjdk-1.8.0.131-11.b12.el7.x86\_64/jre"

HADOOP\_CONF\_DIR="/usr/local/hadoop/etc/hadoop"

– Xml 文件配置格式

<property>

<name>关键字</name>

<value>变量值</value>

<description> 描述 </description>

</property>

官方文档

<http://hadoop.apache.org/>

name value（默认值，可以修改） description

– 核心配置文件 core-site.xml

– fs.defaultFS 文件系统配置参数

– hadoop.tmp.dir 数据目录配置参数,数据根目录，相当于mysql根目录

每一台datanode都有该目录存数据，namenode存储的是数据映射关系

/var/hadoop 给所有主机都创建该文件夹

vim core-site.xml

<configuration>

<property>

<name>fs.defaultFS</name>

<value>hdfs://namenode:9000</value>

//集群时写hdfs://namenode主机名:9000

</property>

<property>

<name>hadoop.tmp.dir</name>

<value>/var/hadoop</value>

</property>

</configuration>

– HDFS配置文件 hdfs-site.xml

vim hdfs-site.xml

<configuration>

<property>

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

<value>namenode:50070</value> //给datanode用

</property>

<property>

<name>dfs.namenode.secondary.http-address</name>

<value>namenode:50090</value> //和namenode在同一台机

</property>

<property>

<name>dfs.replication</name>//切块数据复制几份

<value>2</value>原数据＋复制＝2份，默认是3份

</property>

</configuration>

在datanode中声明namenode是哪台主机

|  |  |
| --- | --- |
| dfs.namenode.name.dir | file://${hadoop.tmp.dir}/dfs/name |

namenode数据存储2种，默认都放在${hadoop.tmp.dir}/dfs/name下，如果不想存在该目录，修改该配置

|  |  |
| --- | --- |
| dfs.datanode.data.dir | file://${hadoop.tmp.dir}/dfs/data |

datanode的数据默认存在本机的${hadoop.tmp.dir}/dfs/data，如果不想存在该目录，修改该配置，还可以用数据恢复和备份

– 节点配置文件slaves 配置所有的datanodes

vim /usr/local/hadoop/etc/hadoop/slaves

node1

node2

node3

5.– 同步配置

– hadoop 所有节点的配置参数完全一样,我们在一台配置好以后,要把配置文件分发到其它所有主机上去

hadoop配置文件特点：

在一台中安装完直接把文件夹拷贝到其他机上，因为配置完全一模一样

6.在 namenode 上执行格式化操作

./bin/hdfs namenode -format

7.在 namenode上启动集群

./sbin/start-dfs.sh

会自动创建一个logs文件夹

./sbin/stop-dfs.sh

jps 每台机输入jps查看自己的角色

在 namenode上运行以下命令：查看集群是否组建成功

./bin/hdfs dfsadmin -report

Live datanodes (3) -----------------------代表成功，3台datanode,哪台没有上那台机排错

排错：

cd logs

.out屏幕输出

.log 日志输出

* [core-default.xml](http://hadoop.apache.org/docs/r2.7.6/hadoop-project-dist/hadoop-common/core-default.xml) 全局核心配置文件
* [hdfs-default.xml](http://hadoop.apache.org/docs/r2.7.6/hadoop-project-dist/hadoop-hdfs/hdfs-default.xml)
* [mapred-default.xml](http://hadoop.apache.org/docs/r2.7.6/hadoop-mapreduce-client/hadoop-mapreduce-client-core/mapred-default.xml)
* [yarn-default.xml](http://hadoop.apache.org/docs/r2.7.6/hadoop-yarn/hadoop-yarn-common/yarn-default.xml)

HDFS可以单独用来做分布式文件系统存储使用，类似ceph

分布式计算框架mapreduce部署

改名：mv mapred-site.xml.template mapred-site.xml

vim mapred-site.xml

<configuration>

<property>

<name>mapreduce.framework.name</name>

<value>yarn</value>

</property>

</configuration>

local自己管理自己 yarn有集群管理者，要遵循集群管理者分配和调度

YARN

管理分布式计算框架mapreduce

vim yarn-site.xml

<configuration>

<property>

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

<value>namenode</value>

//value为resourcemanager所在的主机名，告诉所有的nodemanager，resourcemanager是谁

</property>

<property>

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

<value>mapreduce\_shuffle</value>

//声明你所管理的分布式框架是谁，value为计算框架的名称（开发给的计算框架名称，没有就用官网给的例子mapreduce\_shuffle测试）

</property>

</configuration>

启动服务：sbin/start-yarn.sh

也可以start-all.sh会启动YARN和hdfs

先验证角色 jps

测试集群是否成功：

bin/yarn node -list

• 使用 web 访问hadoop

– namenode web页面(nn01)

– http://192.168.1.10:50070/

– secondory namenode web 页面(nn01)

– http://192.168.1.10:50090/

– datanode web 页面(node1,node2,node3)

– <http://192.168.1.11:50075/>

• 使用 web 访问hadoop

– resourcemanager web页面(nn01)

– http://192.168.1.10:8088/

– nodemanager web 页面(node1,node2,node3)

– <http://192.168.1.11:8042/>

HDFS 基本使用

HDFS 基本命令

./bin/hadoop fs 回车 所支持的shell命令

– ./bin/hadoop fs –ls /

– 对应 shell 命令 ls /

– ./bin/hadoop fs -mkdir /abc

– 对应 shell 命令 mkdir /abc

– ./bin/hadoop fs –rmdir /abc

– 对应 shell 命令 rmdir /abc

• HDFS 基本命令

– ./bin/hadoop fs –touchz /urfile

– 对应 shell 命令 touch /urfile

– ./bin/hadoop fs –cat /urfile

– 对应 shell 命令 cat /urfile

– ./bin/hadoop fs –rm /urfile

– 对应 shell 命令 rm /urfile

集群文件删除一定要写绝对路径

上传（put）下载（get） 本地文件只能上传到集群中

– 上传文件

– ./bin/hadoop fs –put localfile /remotefile

– 下载文件

– ./bin/hadoop fs –get /remotefile

节点管理

hdfs节点管理

• HDFS 增加节点

– 1、启动一个新的系统,禁用 selinux、禁用 firewalld

– 2、设置 ssh 免密码登录

– 3、在所有节点增加新新节点的主机信息 /etc/hosts

– 4、安装 java openjdk 运行环境

– 5、拷贝namnode的 /usr/local/hadoop 到本机

– 6、修改namenode的slaves文件增加该节点

– 7、在该节点启动Datanode

./sbin/hadoop-daemon.sh start datanode （单独启动角色用）

– 8、设置同步带宽,并同步数据 //将新增加的节点均匀分摊其他节点的数据

./bin/hdfs dfsadmin -setBalancerBandwidth 67108864 //设置带宽，网速的一半

./sbin/start-balancer.sh

– 9、查看集群状态

./bin/hdfs dfsadmin -report

• HDFS 修复节点

– 修复节点比较简单,不增加节点基本一致

– 需要注意新节点的 ip 不 主机名 要不损坏节点一致

– 启动服务

./sbin/hadoop-daemon.sh start datanode

– 数据恢复是自动的

– 我们上线以后会自动恢复数据,如果数据量非常巨大,可能需要一定的时间，千万不能手动物理拷贝

• HDFS 删除节点 （危险系数大）

1. 无数据才能删除
2. 查看状态为Decommissioned才能删除
3. 否则会导致整个集群不能用

– 配置NameNode主机上的 hdfs-site.xml

– 增加 dfs.hosts.exclude 配置

<property>

<name>dfs.hosts.exclude</name>

<value>/usr/local/hadoop/etc/hadoop/exclude</value> 创建该文件

</property

– 增加 exclude 配置文件,写入要删除的节点 主机名

vim /usr/local/hadoop/etc/hadoop/exclude

newnode -------------要删除的主机名

– 开始导出数据，将自己的数据均匀分摊到其他主机上

./bin/hdfs dfsadmin -refreshNodes （Decommissioned in Program）

• HDFS 删除节点状态

– 查看状态 ./bin/hdfs dfsadmin -report

– Normal 正常状态

– Decommissioned in Program 数据正在迁移，不能中断

– Decommissioned 数据迁移完成

– 注意:只有当状态变成 Decommissioned 才能 down机下线

连接到要删除的主机上

./sbin/hadoop-daemon.sh stop datanode

grep -p “newnode” -A 14 xxxx.txt5

查询newnode后面的14行

Yarn节点管理

• Yarn 的相关操作

– 由于在 2.x hadoop 引入了 yarn 框架,对于计算节点的操作已经变得非常简单

– 增加节点

在新加节点主机中运行：

sbin/yarn-daemon.sh start nodemanager

– 删除节点

在要删除的节点主机中运行： （在空闲没有计算任务的时候停止服务）

sbin/yarn-daemon.sh stop nodemanager

– 查看节点列表 (Resourcemanager)

在namenode主机上运行：

./bin/yarn node -list

NFS网关

把hdfs虚拟出来的大的硬盘把它变成一个NFS服务器，其他机器可以用这个mount到其他主机上，容量可以无限横向扩容，而且有数据备份

HDFS可以用replication做数据冗余备份，client和NFS在一台机器上，该台机器如果挂了就不能访问后端数据，因为本身不存数据所以无关紧要，做2台高可用就可以了，客户端这台机负责接收前台数据和返回客户端数据

• 特性与注意事项

– HDFS超级用户是不NameNode进程本身具有相同标识的用户,超级用户可以执行任何操作,因为权限检查永远不会为超级用户失败。

– 在非安全模式,运行网关的用户是代理用户

– 在安全模式时,Kerberos keytab中的用户是代理用户

• 调试与日志排错（出错就开调试模式）

– 在配置 NFS 网关过程中经常会碰到各种各样的错误,如果出现了错误,打开调试日志是一个不错的选择

• log4j.property

– log4j.logger.org.apache.hadoop.hdfs.nfs=DEBUG

– log4j.logger.org.apache.hadoop.oncrpc=DEBUG

• 配置代理用户

– 在 namenode 和 nfsgw 上添加代理用户

– 代理用户的 uid gid 用户名 必须完全相同

– 如果因特殊原因客户端的用户和NFS网关的用户 uid不能保持一致需要我们配置 nfs.map 的静态映射关系

– nfs.map

uid 10 100 # Map the remote UID 10 the local UID 100

gid 11 101 # Map the remote GID 11 to the local GID 101

node5 做NFS网关配置

1.配置/etc/hosts（同步到所有主机）

namenode

node1

node2

node3

nfsgw

2.添加用户（namenode、nfsgw两台主机上执行） uid和gid完全相同

groupadd -g 200 nsd1803

useradd -u 200 -g 200 nsd1803

3、在namenode上配置core-site.xml 同步配置到node1-node3,不包括nfsgw

1）./sbin/stop-all.sh 先停止集群

2）vim core-site.xml

<property>

<name>hadoop.proxyuser.nsd1803.groups</name>

<value>\*</value> //允许所有

</property>

<property>

<name>hadoop.proxyuser.nsd1803.hosts</name>

<value>\*</value>

</property>

启动集群

./sbin/start-dfs.sh

1. NFS 网关（以下全是在nfsgw主机上操作）

– 1）、启动一个新的系统,禁用 selinux、禁用 firewalld

– 2）、卸载 rpcbind 、nfs-utils

– 3）、配置 /etc/hosts,添加所有 namenode 和datanode 的主机名不 ip 对应关系

– 4）、安装 java openjdk 的运行环境

– 5）、同步 namenode 的 /usr/local/hadoop 到本机

– 6）、配置 hdfs-site.xml （namenode上操作）

vim hdfs-site.xml

<property>

<name>nfs.exports.allowed.hosts</name>

<value>\* rw</value> //允许所有主机可读写

</property>

– 默认情况下,export可以被任何客户端挂载。为了更好的控制访问,可以设置属性。值字符串为机器名和访问策略,通过空格来分割。机器名的格式可以是单一的主机,Java的正则表达式或者IPv4地址。访问权限使用rw或ro来指定导出目录的读/写或机器只读访问。如果访问策略没被提供,默认为只读的。每个条使用“;”来分割。

<property>

<name>nfs.dump.dir</name>

<value>/var/nfstmp</value>

//临时存储的目录，当上传的整个文件上传完整了才会进行操作

</property>

配置完该属性后要创建 /var/nfstmp 文件夹，mkdir /var/nfstmp

//由nfs的代理用户进行转储，所以该目录对代理用户要有可读写权限

chown 200:200 /var/nfstmp

用户需要更新文件转储目录参数。NFS客户端经常重新安排写操作,顺序的写操作会以随机到达NFS网关。这个目录常用于临时存储无序的写操作。对于每个文件,无序的写操作会在他们积累在内存中超过一定阈值(如。1 mb)被转储。需要确保有足够的空间的目录。例如,如果应用上传10个100M,那么这个转储目录推荐有1GB左右的空间,以便每个文件都发生最坏的情况。只有NFS网关需要在设置该属性后重启。

7）创建数据根目录 /var/hadoop

mkdir /var/hadoop

8）– 设置 /usr/local/hadoop/logs 权限,为代理用户赋予读写执行的权限

setfacl -m u:nsd1803:rwx /usr/local/hadoop/logs

9）、启动服务,以下顺序不能错

– 使用 root 用户启动 portmap 服务

centos6 : rpcbind centos7:portmap

./sbin/hadoop-daemon.sh --script ./bin/hdfs start portmap

– 使用代理用户启动 nfs3

su -l nsd1803

./sbin/hadoop-daemon.sh --script ./bin/hdfs start nfs3

客户端测试：

客户端安装nfs-utils

mount -t nfs -o vers=3,proto=tcp,nolock,noacl,noatime,sync 192.168.1.15:/ /mnt

vers=3 目前NFS只支持V3版本

proto=tcp 仅使用TCP作为传输协议

nolock不支持锁NLM

noatime 禁用时间更新 数据切块完只修改时间啥也不干就到下一步了，浪费性能（优化）

atime mtime ctime

stat /etc/passwd

Access time最近访问：2018-08-02 17:04:43.299039430 +0800

Modify time最近更改：2018-08-02 17:04:43.098037406 +0800

内容或属性发生变化，文件Md5值变化了一定会更改

Change time最近改动：2018-08-02 17:04:43.123037658 +0800

属性发生变化

sync 不支持随机写，只支持顺序写，会导致不可预测的吞吐量，未指定同步选项可能会导致上传大文件时出现不可靠的行为

查看注册服务

rpcinfo -p 192.168.1.15

查看共享目录

showmount -e 192.168.1.15

**1、Namenode**

**它是Hadoop 中的主服务器，管理文件系统名称空间和对集群中存储的文件的访问。**

**2、Datanode**

**它负责管理连接到节点的存储（一个集群中可以有多个节点）。每个存储数据的节点运行一个 datanode 守护进程。**

**3、secondaryNameNode**

**它不是 namenode 的冗余守护进程，而是提供周期检查点和清理任务。 出于对可扩展性和容错性等考虑，我们一般将SecondaryNameNode运行在一台非NameNode的机器上。**

**4、ResourceManager**

**负责调度 DataNode上的工作。每个 DataNode有一个NodeManager，它们执行实际工作。**

**5、NodeManager**

**负责执行ResourceManager分发的任务**

Zookeeper

• zookeeper 是什么?

ZooKeeper是一个分布式的,开放源码的分布式应用程序协调服务

• ZooKeeper能干什么哪?

ZooKeeper是用来保证数据在集群间的事务性一致

冲突问题使用锁机制，同时写有锁竞争，先拿到锁的才能操作，没有拿到锁的需要等待

– 集群分布式锁

– 集群统一命名服务

– 分布式协调服务

• zookeeper 角色与特性

– Leader: 负责写 一个集群只有一个leader

– 接受所有Follower的提案请求并统一协调发起提案的投票,负责不所有的Follower进行内部的数据交换

– Follower: 负责读

– 直接为客户端服务并参与提案的投票,同时不Leader进行数据交换

– Observer:

– 直接为客户端服务但并不参与提案的投票,同时也与Leader进行数据交换

• zookeeper 角色不选举

– 服务在启动的时候是没有角色的 (LOOKING)

– 角色是通过选举产生的

– 选举产生一个 leader,剩下的是 follower

– 选举 leader 原则:

– 集群中超过半数机器投票选择leader.

– 假如集群中拥有n台服务器,那么leader必须得到n/2+1台服务器投票

• zookeeper 角色与选举

（集群最好配置奇数台，偶数台浪费）

– 如果 leader 死亡,从新选举 leader

– 如果死亡的机器数量达到一半(包含所有角色),集群挂起

– 如果无法得到足够的投票数量,就重新发起投票,如果参不投票的机器不足 n/2+1 集群停止工作

– 如果 follower 死亡过多,剩余机器不足 n/2+1 集群也会停止工作

– observer 不计算在投票总设备数量里面

5台leader ＋follower

3台observer 不参与投票，只需要3票通过就可以

• zookeeper 可伸缩扩展性原理不设计

– leader 所有写相关操作

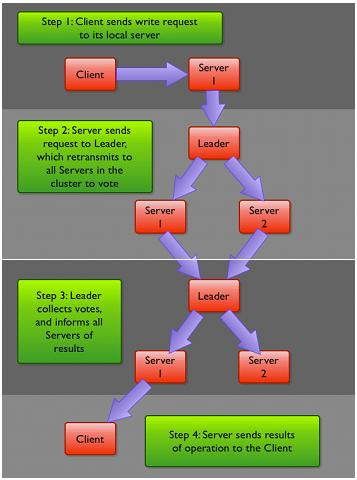
– follower 读操作与响应leader提议

– 在Observer出现以前,ZooKeeper的伸缩性由Follower来实现,我们可以通过添加Follower节点的数量来保证ZooKeeper服务的读性能。但是随着Follower节点数量的增加,ZooKeeper服务的写性能受到了影响。为什么会出现这种情况?在此,我们需要首先了解一下这个"ZK服务"是如何工作的。

• zookeeper 可伸缩扩展性原理与设计

– 客户端提交一个请求,若是读请求,则由每台Server的本地副本数据库直接响应。若是写请求,需要通过一致性协议(Zab)来处理

– Zab协议规定:来自Client的所有写请求,都要转发给ZK服务中唯一的Leader,由Leader根据该请求发起一个提案Proposal。然后,其他的Server对该提案Proposal进行投票Vote。之后,Leader对Vote进行收集,当Vote数量过半时Leader会吐所有的Server发送一个通知消息。最后,当Client所连接的Server收到该消息时,会把该操作更新到内存中并对Client的写请求做出回应



ZooKeeper 服务器在上述协议中实际扮演了两个职能。

它们一方面从客户端接受连接不操作请求,另一方面对操作结果进行投票。这两个职能在 ZooKeeper集群扩展的时候彼此制约

– 从Zab协议对写请求的处理过程中我们可以发现,增加follower的数量,则增加了对协议中投票过程的压力。因为Leader节点必须等待集群中过半Server响应投票,于是节点的增加使得部分计算机运行较慢,从而拖慢整个投票过程的可能性也随之提高,随着集群变大,写操作也会随之下降

– 所以,我们不得不,在增加Client数量的期望和我们希望保持较好吞吐性能的期望间进行权衡。要打破这一耦合关系,我们引入了不参与投票的服务器,称为Observer。 Observer可以接受客户端的连接,并将写请求转发给Leader节点。但是,Leader节点不会要求 Observer参加投票。相反,Observer不参与投票

过程,仅仅在上述第3步那样,和其他服务节点一起得到投票结果

– Observer的扩展,给 ZooKeeper 的可伸缩性带来了全新的景象。我们现在可以加入很多 Observer 节点,而无须担心严重影响写吞吐量。但他并非是无懈可击的,因为协议中的通知阶段,仍然不服务器的数量呈线性关系。但是,这里的串行开销非常低。因此,我们可以讣为在通知服务器阶段的开销不会成为瓶颈

– Observer提升读性能的可伸缩性

– Observer提供了广域网能力

– Observer除了不能投票其他都一样

– Observer的扩展,给 ZooKeeper 的可伸缩性带来了

全新的景象。我们现在可以加入很多 Observer 节点,而无须担心严重影响写吞吏量。但他并非是无懈可击的,因为协议中的通知阶段,仍然不服务器的数量呈线性关系。但是,这里的串行开销非常低。因此,我们可以讣为在通知服务器阶段的开销并不会成为瓶颈

– Observer提升读性能的可伸缩性

– Observer提供了广域网能力

ZK 集群的安装配置

只能坏一台，2台就没有高可用了

1. 配置/etc/hosts ,所有集群主机可以互相ping通

vim /etc/hosts

192.168.1.20 zookeeper1

192.168.1.21 zookeeper1

192.168.1.22 zookeeper1

192.168.1.23 zookeeper1

2、安装 openjdk 环境

3、解压zookeeper源码文件，并创建配置文件

mv zookeeper /usr/local/zookeeper

1. 修改配置文件

mv zoo\_sample.cfg zoo.cfg

vim zoo.cfg

server.1=zookeeper1:2888:3888

server.2=zookeeper2:2888:3888

server.3=zookeeper3:2888:3888

server.4=zookeeper4:2888:3888:observer

– server.（myid）

2、– 创建 datadir 指定的目录（配置文件中指定该目录，为数据存放的目录）

– mkdir /tmp/zookeeper

1. 为每一台服务器创建一个ID（类似mysql的serverid）

echo 4 > /tmp/zookeeper/myid

(和上面配置文件中指定的主机和id要对应上）

– 在对应主机目录下创建 id 对应的主机名的 myid 文件

– 关于myid文件:

– myid文件中只有一个数字

– 注意,请确保每个server的myid文件中id数字不同

– server.id 中的 id 不 myid 中的 id 必须一致

– id的范围是1~255

1. 在所有集群节点服务器上手动启动服务

/usr/local/zookeeper/bin/zkServer.sh start 启动服务

/usr/local/zookeeper/bin/zkServer.sh status 查看角色

或者 { echo 'stat';yes; }|telnet 192.168.4.10 2181 查看状态

拷贝一台主机的zookeeper文件夹到其他集群主机上

<http://zookeeper.apache.org/doc/r3.4.10/zookeeperAdmin.html>

zookeeper 官方管理文档

zookeeper1]# telnet zookeeper3 2181 用该命令在一台查询其他机的状态

{echo ruok;sleep 0.2;} | telnet zookeeper3 2181 | grep imok

echo $?

写脚本测试

查询单台主机的状态

循环查询这几台主机的状态

kafka 集群

分布式的消息系统

为什么要使用 kafka

– 解耦、冗余、提高扩展性、缓冲

– 保证顺序,灵活,削峰填谷

– 异步通信

• kafka 角色与集群结构

– producer:生产者,负责发布消息

– consumer:消费者,负责读取处理消息

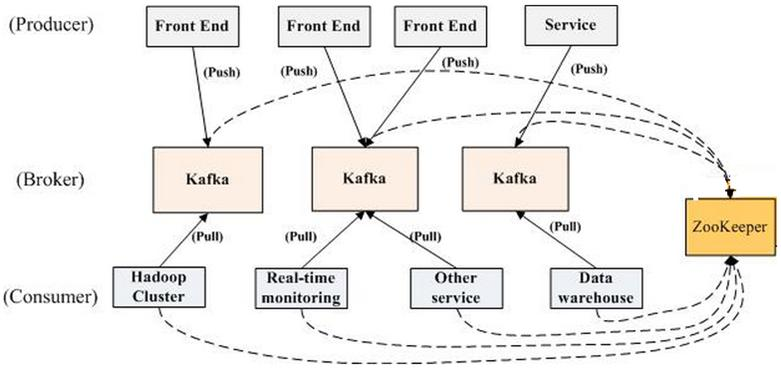
– topic:消息的类别

– Parition:每个Topic包含一个或多个Partition.

– Broker:Kafka集群包含一个戒多个服务器

– Kafka通过Zookeeper管理集群配置,选举leader

kafka 角色与集群结构



• kafka 集群的安装配置

– kafka 集 群 的 安 装 配 置 是 依 赖 zookeeper 的 , 搭 建

kafka 集群之前,首先请创建好一个可用 zookeeper集群

– 安装 openjdk 运行环境

– 分发 kafka 拷贝到所有集群主机

– 修改配置文件

– 启动与验证

• kafka 集群的安装配置

配置etc/hosts

192.168.1.30 kafka1

192.168.1.31 kafka2

192.168.1.32 kafka3

192.168.1.20 zookeeper1

192.168.1.21 zookeeper2

192.168.1.22 zookeeper3

192.168.1.23 zookeeper4

1、vim server.properties

broker.id=0 // 每台服务器的broker.id都不能相同

zookeeper.connect=zookeeper1:2181,zookeeper2:2181,zookeeper3:2181

– zookeeper 集群地址,不用都列出,写一部分即可

1. 拷贝/usr/local/kafka 到集群其他主机 3台
2. – 在所有主机启动服务

/usr/local/kafka/bin/kafka-server-start.sh -daemon /usr/local/kafka/config/server.properties

4、验证

– jps 命令应该能看到 kafka 模块

– netstat 应该能看到 9092 在监听

1. • 集群验证不消息发布

创建一个 topic

./bin/kafka-topics.sh --create --partitions 1 --replication-factor 1 --zookeeper zookeeper3:2181 --topic nsd1803

生产者

./bin/kafka-console-producer.sh --broker-list zookeeper2:9092 --topic nsd1803

消费者

1. ./bin/kafka-console-consumer.sh --bootstrap-server zookeeper1:9092 --topic nsd1803
2. 查看topic

bin/kafka-topics.sh --describe --zookeeper zookeeper1:2181

查看topic，此时的leader kafka为129，repicas：集群里有三个kafka，Isr：正常使用的kafka

日志：/tmp/kafka-logs

/usr/local/kafka/logs/server.log

NameNode 高可用

为什么 NameNode 需要高可用

NameNode 是 HDFS 的核心配置,HDFS 又是Hadoop 的核心组件,NameNode 在 Hadoop 集群中至关重要,NameNode机器宕机,将导致集群不可用,如果NameNode 数据丢失将导致整个集群的数据丢失,而 NameNode 的数据的更新又比较频繁,实现 NameNode 高可用势在必行

– 官方提供了两种解决方案

– HDFS with NFS

– HDFS with QJM

– 两种方案异同

NFS QJM

NN NN

ZK ZK

ZKFailoverController ZKFailoverController

NFS JournalNode 存储位置

HA 方案对比

– 都能实现热备

– 都是一个active NN 和一个 standby NN

– 都使用Zookeeper 和 ZKFC 来实现自动失效恢复

– 失效切换都使用 fencing 配置的方法来 active NN

– NFS 数据数据共享变更方案把数据存储在共享存储里面,我们还需要考虑 NFS 的高可用设计

– QJM 不需要共享存储,但需要让每一个 DN 都知道两个 NN 的位置,并把块信息和心跳包发送给active和standby这两个 NN，不需要共享存储，但是传输量变成2倍了，配置2个namenode（一主一备）

• NameNode 高可用方案 (QJM)

– 为了解决 NameNode 单点故障问题,Hadoop 给出了 HDFS 的高可用HA方案:HDFS 通常由两个NameNode组成,一个处于 active 状态,另一个处于

standby 状态。Active NameNode对外提供服务,比如处理来自客户端的 RPC 请求,而 StandbyNameNode 则不对外提供服务,仅同步 ActiveNameNode 的状态,以便能够在它失败时进行切换。

• Secondary NameNode

– 定期合并 fsimage（数据块的映射信息） 和fsedits（数据变更日志）,推送给NameNode

datanode 给两台datanode 发fsimage，不再需要Secondary NameNode，

变更日志写在JNS上，NN1往JNS上写，NN2往JNS上读，为了实现JNS高可用，也要做多台服务器，如果NN1(active)挂了，NN2（standby）先把JNS上的读过来再把自己变成active

• NameNode 高可用架构

– 一个典型的HA集群,NameNode会被配置在两台独立的机器上,在任何时间上,一个NameNode处于活动状态,而另一个NameNode处于备份状态,活劢状

态的NameNode会响应集群中所有的客户端,备份状态的NameNode只是作为一个副本,保证在必要的时候提供一个快速的转移。NameNode 高可用

• NameNode 高可用架构

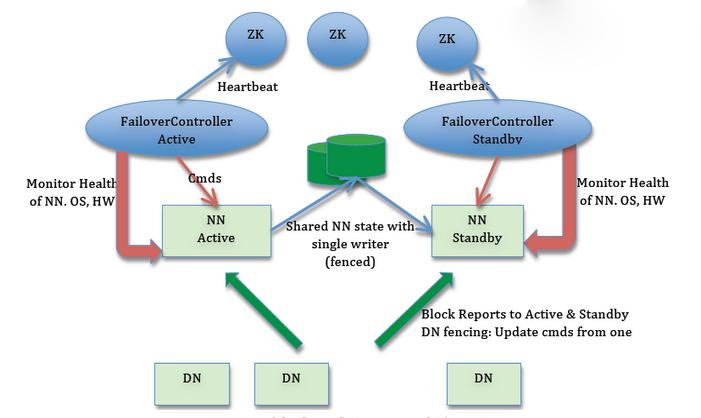
– 为了让Standby Node不Active Node保持同步,这两个Node都不一组称为JNS的互相独立的迚程保持通信(Journal Nodes)。当Active Node上更新了namespace,它将记录修改日志发送给JNS的多数派。Standby noes将会从JNS中读取这些edits,并持续关注它们对日志的变更。Standby Node将日志变更应用在自己的namespace中,当failover发生时,Standby将会在提升自己为Active之前,确保能够从JNS中读取所有的edits,即在failover发生之前Standy持有的namespace应该不Active保持完全同步。

– NameNode 更新是很频繁的,为了的保持主备数据的一致性,为了支持快速failover,Standby node持有集群中blocks的最新位置是非常必要的。为了达到这一目的,DataNodes上需要同时配置这两个Namenode的地址,同时和它们都建立心跳链接,并把block位置发送给它们NameNode 高可用

– 还有一点非常重要,任何时刻,只能有一个ActiveNameNode,否则将会导致集群操作的混乱,那么两个NameNode将会分别有两种丌同的数据状态,可能

会导致数据丢失,戒者状态异常,这种情冴通常称为“split-brain”(脑裂,三节点通讯阻断,即集群中丌同的Datanode 看到了不同的Active NameNodes)。

对于JNS而言,任何时候只允讲一个NameNode作为writer;在failover期间,原来的Standby Node将会接管Active的所有职能,并负责吐JNS写入日志记录,这中机制阻止了其他NameNode基于处于Active状态的问题。



如果看到两台active，先干掉一个

部署

ALL: 配置 /etc/hosts

192.168.1.10 nn01

192.168.1.20 nn02

192.168.1.11 node1

192.168.1.12 node2

192.168.1.13 node3

ALL: 除了 zookeeper 其他 hadoop ，kafka 服务全部停止

ALL: 初始化 hdfs 集群，删除 /var/hadoop/\*

NN2: 关闭 ssh key 验证，部署公钥私钥

StrictHostKeyChecking no

scp nn01:/root/.ssh/id\_rsa /root/.ssh/

scp nn01:/root/.ssh/authorized\_keys /root/.ssh/

修改配置文件

• core-site.xml 增加以下内容：

<property>

<name>fs.defaultFS</name>

<value>hdfs://mycluster</value> //mycluster随便取一个名做为组名，把两个NN放入这个组中

</property>

<property>

<name>hadoop.tmp.dir</name>

<value>/var/hadoop</value>

</property>

<property>

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

<value>zookeeper1:2181,zookeeper2:2181,zookeeper3:2181</value>//建议写3个，防止有一台坏了

</property>

• hdfs-site.xml 只留下以下内容，删除secondaryname内容

– secondarynamenode 在高可用里面没有用途,这里把他关闭

– namenode 在后面定义

<property>

<name>dfs.replication</name>

<value>2</value>

</property>

– <!-- 指定定hdfs的nameservices名称为mycluster -->

<property>

<name>dfs.nameservices</name>

<value>mycluster</value>

</property>

– 指定集群的两个 NaneNode 的名称分别为nn1,nn2，变量，不是主机名

<property>

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

<value>nn1,nn2</value>

</property>

– 配置nn1,nn2的rpc通信端口，因为要让所有的datanode知道nn1和nn2的地址

<property>

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

<value>node1:8020</value> //node1是真正的主机名：端口

</property>

<property>

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

<value>node2:8020</value>

</property>

– 配置nn1,nn2的http通信端口

<property>

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

<value>namenode:50070</value>

</property>

<property>

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

<value>namenode1:50070</value>

</property>

– 指定namenode元数据存储在journalnode中的路径 （secondaryname）

<property>

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

<value>qjournal://node1:8485;node2:8485;node3:8485/mycluster</value>

</property>

– 指定HDFS客户端连接active namenode的java类 ZKFC

<property>

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

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

</property>

– 配置隔离机制为 ssh

<property>

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

<value>sshfence</value>

</property>

– 指定私钥的位置

<property>

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

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

</property>

– 开启自动故障转移

<property>

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

<value>true</value>

</property>

NameNode 高可用验证

• 同步数据不高可用验证

– 同步配置到所有集群机器

– 在其中一台初始化 zookeeper 集群

bin/hdfs zkfc -formatZK

– 在定义的节点启动 journalnode

sbin/hadoop-daemon.sh start journalnode

– 所有节点都要启动NameNode 高可用

• 同步数据不高可用验证

– 在其中一台 namenode 上执行格式化命令

bin/hdfs namenode –format

– 注意是格式化其中一台,格式化以后把数据目录拷贝到另一台

– 初始化 JournalNode

./bin/hdfs namenode –initializeSharedEditsNameNode 高可用

• 同步数据不高可用验证

– 停止 JournalNode

sbin/hadoop-daemon.sh stop journalnode

– 启动 dfs

./sbin/start-dfs.sh startNameNode 高可用

• 同步数据不高可用验证

– 验证配置

bin/hadoop dfsadmin –report

– 查看集群状态

bin/hdfs haadmin -getServiceState nn1

bin/hdfs haadmin -getServiceState nn2NameNode 高可用

• 同步数据不高可用验证

– 测试文件读写不 namenode 故障转移

bin/hadoop fs -ls hdfs://mycluster/

bin/hadoop fs -mkdir hdfs://mycluster/input

bin/hadoop fs -put \*.txt hdfs://mycluster/input

bin/hadoop

jar

share/hadoop/mapreduce/hadoop-

mapreduce-examples-2.7.3.jar

wordcount

hdfs://mycluster/input hdfs://mycluster/output

– 关闭主Namenode

bin/hdfs haadmin -getServiceState nn1

bin/hdfs haadmin -getServiceState nn2

bin/hadoop fs -cat hdfs://mycluster/output/\*

ResourceManager

Yarn 高可用

• ResourceManager 高可用

– RM 的高可用原理和 NN 是一样的,需要依赖 ZK 来实现,这里就不重复了,只给出配置文件的关键部分,感兴趣的同学可以自己学习和测试

– yarn.resourcemanager.hostname

– 同理因为使用集群模式,该选项应该关闭

• yarn-site.xml 配置

<property>

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

<value>true</value>

</property>

<property>

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

<value>rm1,rm2</value>//rm（resourcemanager）变量和nn变量一样

</property>

<property>

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

<value>true</value>

</property>

<property>

<name>yarn.resourcemanager.store.class</name>

<value>org.apache.hadoop.yarn.server.resourcemanager.re

covery.ZKRMStateStore</value>//在zookeeper上存储信息的类

</property>

<property>

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

<value>zookeeper1:2181,zookeeper2:2181,zookeeper3:2181</value>//zookeeper服务器

</property>

<property>

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

<value>yarn-ha</value>

</property>

<property>

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

<value>namenode</value>

</property>

<property>

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

<value>namenode1</value>

</property>

• ResourceManager 高可用

– 记动集群

sbin/start-yarn.sh start

– 查看集群状态

bin/yarn rmadmin -getServiceState rm1

bin/yarn rmadmin -getServiceState rm2

NN1: 配置 core-site.xml

<configuration>

<property>

<name>fs.defaultFS</name>

<value>hdfs://nsdcluster</value>

</property>

<property>

<name>hadoop.tmp.dir</name>

<value>/var/hadoop</value>

</property>

<property>

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

<value>node1:2181,node2:2181,node3:2181</value>

</property>

<property>

<name>hadoop.proxyuser.nsd1803.groups</name>

<value>\*</value>

</property>

<property>

<name>hadoop.proxyuser.nsd1803.hosts</name>

<value>\*</value>

</property>

</configuration>

配置 hdfs-site.xml

<configuration>

<property>

<name>dfs.replication</name>

<value>2</value>

</property>

<property>

<name>dfs.nameservices</name>

<value>nsdcluster</value>

</property>

<property>

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

<value>nn1,nn2</value>

</property>

<property>

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

<value>nn01:8020</value>

</property>

<property>

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

<value>nn02:8020</value>

</property>

<property>

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

<value>nn01:50070</value>

</property>

<property>

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

<value>nn02:50070</value>

</property>

<property>

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

<value>qjournal://node1:8485;node2:8485;node3:8485/nsdcluster</value>

</property>

<property>

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

<value>/var/hadoop/journal</value>

</property>

<property>

<name>dfs.client.failover.proxy.provider.nsdcluster</name>

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

</property>

<property>

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

<value>sshfence</value>

</property>

<property>

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

<value>/root/.ssh/id\_rsa</value>

</property>

<property>

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

<value>true</value>

</property>

</configuration>

yarn-site.xml 配置文件

<configuration>

<!-- Site specific YARN configuration properties -->

<property>

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

<value>mapreduce\_shuffle</value>

</property>

<property>

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

<value>true</value>

</property>

<property>

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

<value>rm1,rm2</value>

</property>

<property>

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

<value>true</value>

</property>

<property>

<name>yarn.resourcemanager.store.class</name>

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

</property>

<property>

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

<value>node1:2181,node2:2181,node3:2181</value>

</property>

<property>

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

<value>yarn-ha</value>

</property>

<property>

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

<value>nn01</value>

</property>

<property>

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

<value>nn02</value>

</property>

</configuration>

#-----------------------------------------------------#

初始化启动集群

ALL: 所有机器

nodeX： node1 node2 node3

NN1: nn01

NN2: nn02

#-----------------------------------------------------#

ALL: 同步配置到所有集群机器

NN1: 初始化ZK集群 ./bin/hdfs zkfc -formatZK （Successfully）

nodeX(所有datanode): 启动 journalnode 服务

./sbin/hadoop-daemon.sh start journalnode

（启动完查看角色：jps:journalnode）

启动完所有再格式化：

NN1: 格式化 ./bin/hdfs namenode -format （Successfully）

NN2: 数据同步到本地 /var/hadoop/dfs

NN2和NN1的这个/var/hadoop/dfs目录结构要一模一样 copy

NN1: 初始化 JNS

./bin/hdfs namenode -initializeSharedEdits （Successfully）

nodeX: 要用集群管理，所以停止 journalnode 服务 jps 看不见该角色

./sbin/hadoop-daemon.sh stop journalnode

#-----------------------------------------------------#

启动集群

NN1: ./sbin/start-all.sh

NN2: ./sbin/yarn-daemon.sh start resourcemanager

启动完回到NN1，查看集群状态

./bin/hdfs haadmin -getServiceState nn1 active

./bin/hdfs haadmin -getServiceState nn2 standby

./bin/yarn rmadmin -getServiceState rm1

./bin/yarn rmadmin -getServiceState rm2

./bin/hdfs dfsadmin -report

./bin/yarn node -list

NN1访问集群：

./bin/hadoop fs -ls /

./bin/hadoop fs -mkdir hdfs://nsdcluster/input

./bin/hadoop fs -ls hdfs://nsdcluster/input //标准全路径

验证高可用，关闭 active namenode

./sbin/hadoop-daemon.sh stop namenode

./sbin/yarn-daemon.sh stop resourcemanager

nn1关闭，nn2变成active

恢复节点

./sbin/hadoop-daemon.sh start namenode

./sbin/yarn-daemon.sh start resourcemanager

nn1启动后nn2还是active,nn1变成standby