**zookeeper**

zookeeper

zookeeper是什么

zookeeper是一个分布式的，开放源代码的分布式应用程序协调服务（协调执行顺序）

zookeeper能干什么

zookeeper是用来保证数据在集群间的事务性一致

zookeeper应用场景

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：

接受所有follower提案请求并统一协调发起提案的投票，负责域所有的follower进行内部的数据交换（老板负责制定员工的工作方向）

follower:

直接为客户端服务并参与提案的投票，同时与leader进行数据交换（经理负责安排活儿和向老板汇报）

observe

直接为客户端服务但并不参提案的投票，同时也与leader进行数据交换（员工直接干具体的活）

zookeeper角色与选举

服务在启动的时候没有角色的（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

如果leader死亡，重新选举leader

如果死亡的集群数量达到一半，集群挂起

如果无法得到足够的投票数量，就重新发起投票，如果参与投票的机器不足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

zookeeper可伸缩性原理与设计

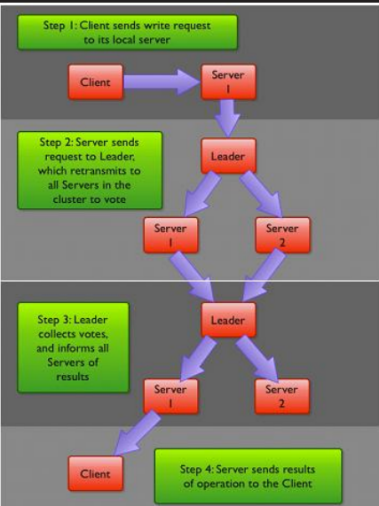
leader所有写相关操作

follower读操作与相应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服务的读写性能。但是随着follokwer节点数量的增加，zookeeper服务的读写性能受到了影响。为什么会出现这种情况？需要首先了解一下这个“ZK”服务是如何工作的。

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。如果是写请求，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proposal。然后，其他的server对该proposal进行vote。之后，leader会向所有的server发送一个通知信息。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的写请求作出回应



zookeeper服务器在上述协议中实际扮演了两个职能，接收客户端请求和参与投票选举。它们一方面从客户端接收连接与操作请求，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国版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，引入了不参与投票的服务器，称为observer。observer可以接手客户端的连接，并将写请求转发给leader节点。但是，Leader节点不会要求observer参与投票。相反，observer不参与投票过程，仅仅在第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。现在可以加入很多observer节点，而无需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，仍然与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，可以认为在通知服务器阶段的开销不会成为瓶颈

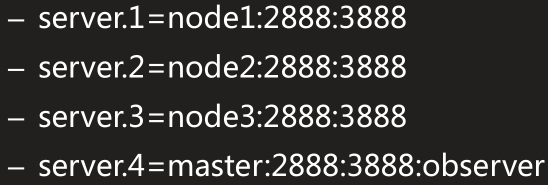
observer提升读性能的可伸缩性

observer提供了广域网能力

ZK集群的安装配置

1. 安装openjdk环境
2. 解压创建配置文件
3. 设置集群机器id、ip、port
4. 拷贝分发到所有集群节点
5. 创建目录和myid文件
6. 启动服务
7. 查看状态

zoo.cfg



zoo.cfg集群的安装配置

创建datadir指定的目录

mkdir /tmp/zookeeper

在目录下创建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与myid中的id必须一致

id的范围是1~255

ZK集群的安装配置

启动集群，查看验证

在所有集群节点执行

/usr/local/zk/bin/zkServer.sh start

查看角色

/usr/local/zk/bin/zkServer.sh status

ZK实验

搭建zookeeper集群

添加observer

查找leader

模拟leader故障

模拟follower故障

故障恢复

**kafka集群**

kafka是什么

kafka是由linkedin开发的一个分布式的消息系统

kafka是由scala编写

kafka是一种消息的中间件

为什么使用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，负责发布信息

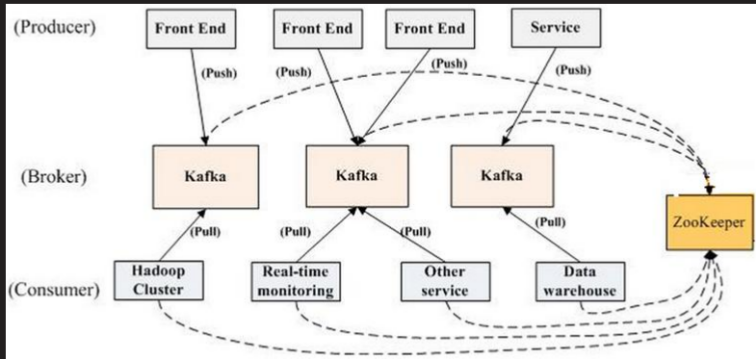
consumer：消费者，负责读取处理信息

topic：消息的类别

parition：每个topic包含一个或多个partition

broker：kafka集群包含一个或多个服务器

kafka通过zookeeper管理集群配置，选举leader



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

在所有主机启动服务

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

验证

jps命令能看到kafa模块

netstat能看到9092在监听

server.properties

broker.id

每台服务器的broker.id都不能相同

zookeeeper.connect

zookeeper集群地址，不用都列出，写一部分即可

集群验证与消息发布

创建一个topic

./bin/kafka-topics.sh --create --partitions 2 --replication-

factor 2 --zookeeper node3:2181 --topic mymsg

查看已经存在的topic

./bin/kafka-topics.sh --list --zookeeper node2:2181

查看刚刚创建的topic

./bin/kafka-topics.sh --describe --zookeeper node1:2181 --

topic mymsg

在两个终端里面，生产者发布消息，消费者读取消息

生产者

./bin/kafka-console-producer.sh --broker-list master:9092,node1:9092 --topic mymsg

消费者

./bin/kafka-console-consumer.sh --bootstrap-server node2:9092,node3:9092 --topic mymsg

--from-beginning表示从开始读取消息

**hadoop高可用**

namenode高可用

为什么namenode需要高可用

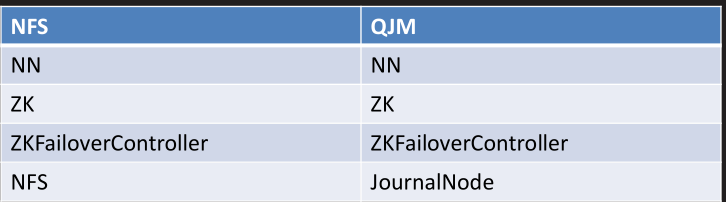
namenode是HDFS的核心配置，HDFS又是hadoop的核心组件，namenode在hadoop集群中至关重要，namenode机器宕机，将导致集群不可用，如果namenode的数据丢失将导致整个集群的数据丢失，而namenode的数据的更新又比较平凡，实现namenode高可用势在必行

官方提供了两种解决方案

HDFS with NFS

HDFS with QJM

两种方案的异同



HA方案对比

都能实现热备

都是一个active NN和一个standby NN

都能使用zookeeper和ZKFC来实现自动失效恢复

失效切换都是使用fencing配置的方法来active NN

NFS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

namenode高可用方案（QJM）

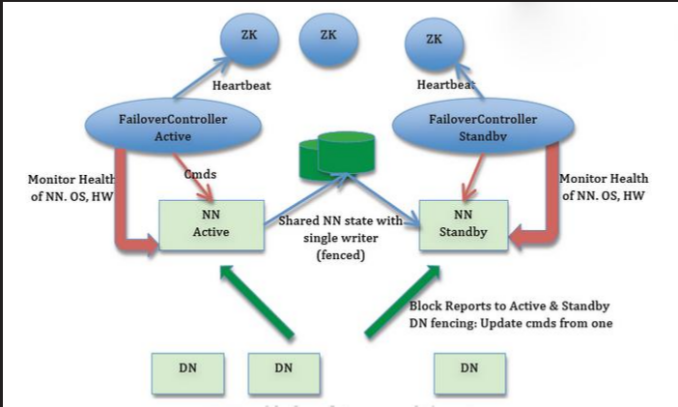
为了解决namenode单点故障问题，hadoop给出了HDFS的高可用HA方案 ：HDFS通常由两个namenode组成，一个处于active状态，另一个处于standby状态。active namenode对外提供服务，比如处理来自客户端的RPC请求，而standby namenode则不对外提供服务，仅同步active namenode的状态，以便能够在它失败时进行切换

一个典型的HA集群，namenode会被配置阿紫两台独立的机器上，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，活动状态的namenmode会相应集群中所有的客户端，备份的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

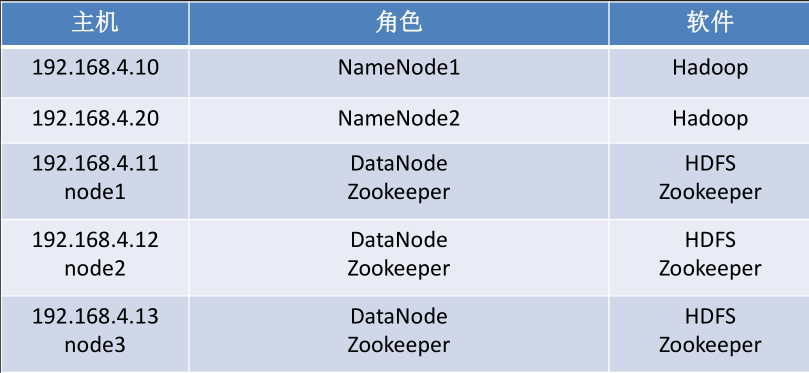
为了让standby node与active node保持同步，这两个node都与一组称为JNS的互相独立的进程保持通信。当active node上更新了namespace，它将记录修改日志发送给JNS的多数派。standby nodes将会从JNS中读取这些edits，并持续关注它们对日志的变更。standby node将日志变更应用在自己的namespace中，当failover发生时，standby将会在提升自己为active之前，确保能够从JNS中读取所有的edits，即在failover发生之前standy持有的namespace应该与active保持完全同步。

namenode更新很频繁，为了保持主备数据的一致性，为了支持快读failover，stabndby 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

还有一点非常重要，任何时刻，只能有一个active namenode，否则将会分别有两种不同的数据状态，可能会导致数据丢失，或者状态异常，这种情况通常称为”split-brain”（脑裂，三节点通信阻断，即集群中不同的datanode看到了不同的active namenodes）。对于JNS而言，任何时候只允许一个namenode作为writer;在failover期间，原来的standby node将会接管active的所有职能，并负责向JNS写入日志记录，这种机制阻止了其他Namenode基于处于active状态的问题



系统规划



core-site.xml

<property>

<name>fs.defaultFS</name>

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

</property>

<property>

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

<value>/var/hadoop</value>

</property>

<property>

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

<value>node2:2181,node3:2181,node4:2181</value>

</property>

hdfs-site.xml

<property>

<name>dfs.replication</name>

<value>1</value>

</property>

secondartnamenode在高可用里面没有用途，这里把他关闭

namenode在后面定义

<!--指定hdfs的nameservices名称为mycluster>

<property>

<name>dfs.nameservices</name>

<value>mycluster</value>

</property>

指定集群的两个namenode的名称分别为nn1，nn2

<property>

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

<value>nn1,nn2</value>

</property>

配置nn1,nn2的rpc通信端口

<property>

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

<value>node1:8020</value>

</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>node1:50070</value>

</property>

<property>

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

<value>node2:50070</value>

</property>

指定namenode元数据存储在journalnode中的路径

<property>

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

<value>qjournal://node3:8485;node4:8485/mycluster</value>

</property>

指定journalnode日志文件存储的路径

<property>

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

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

</property>

指定HDFS客户端连接active namenode的java类

<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>

指定密钥的位置

<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>

同步数据与高可用验证

同步配置到所有集群主机

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

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 –initializeSharedEdits

停止journalnode

sbin/hadoop-daemon.sh stop journalnode

启动dfs

./sbin/start-dfs.sh start

验证配置

bin/hadoop dfsadmin -report

查看集群状态

bin/hdfs haadmin -getServiceState nn01

bin/hdfs haadmin -getServiceState nn02

测试文件读写与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的高可用原理与NN是一样的，需要依赖ZK来实现，这里就不重复了，给出相关配置

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>

</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>node2:2181,node3:2181,node4:2181</value>

</property>

<property>

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

<value>yarn-ha</value>

</property>

<property>

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

<value>node1</value>

</property>

<property>

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

<value>node2</value>

</property>

resourcemanager高可用

启动集群

sbin/start-yarn.sh start

查看集群状态

bin/yarn rmadmin -getServiceState rm1

bin/yarn rmadmin -getServiceState rm2