

# Linux 大数据

**NSD HADOOP**

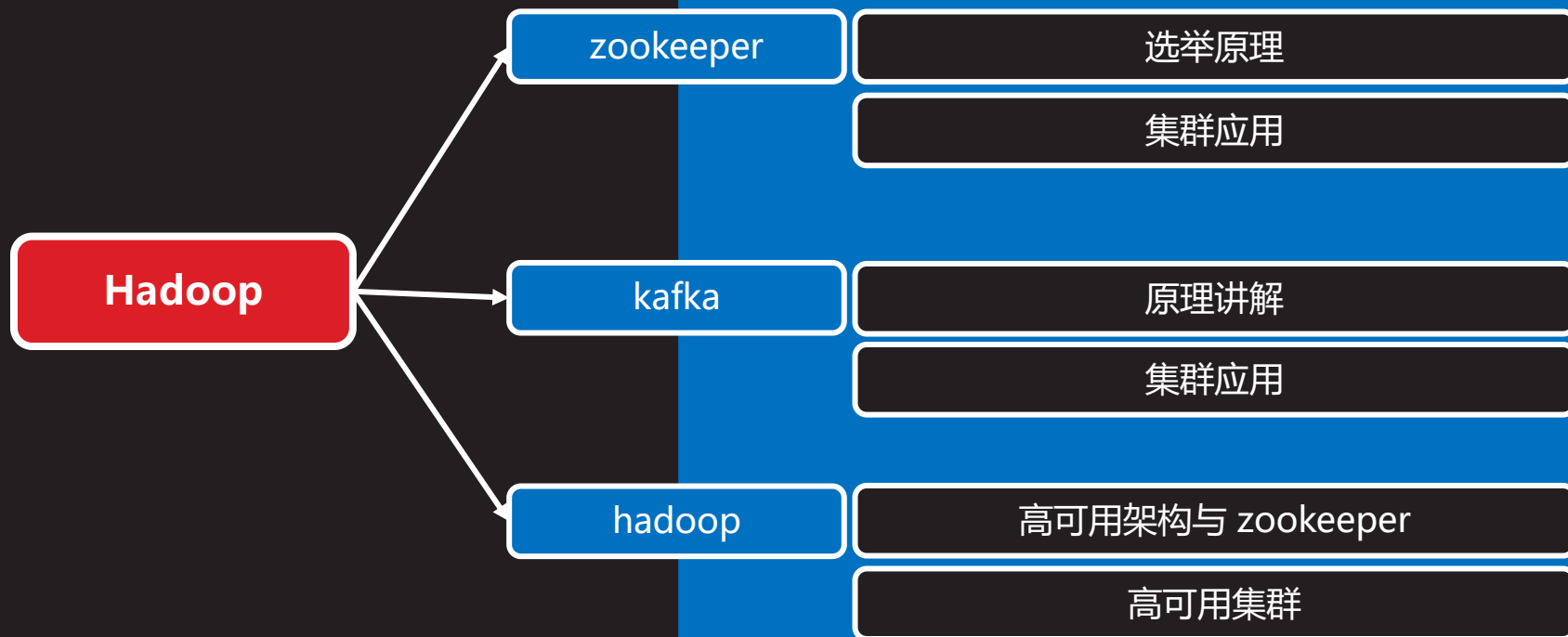
**DAY03**

# 内容

上午	09:00 ~ 09:30	课程回顾
	09:30 ~ 10:20	Zookeeper 集群
	10:30 ~ 11:20	Kafka 集群
	11:30 ~ 12:00	组合实验
下午	14:00 ~ 14:50	Namenode 高可用
	15:00 ~ 15:50	
	16:10 ~ 17:00	Resourcemanager高可用
	17:10 ~ 18:00	总结和答疑



# 大数据



# zookeeper

---

# 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



# zookeeper

- 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

- zookeeper 角色与特性
  - Leader :
    - 接受所有Follower的提案请求并统一协调发起提案的投票，负责与所有的Follower进行内部的数据交换
  - 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
  - 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

- zookeeper 角色与选举
  - 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

- zookeeper 角色与选举
  - 如果 leader 死亡，从新选举 leader
  - 如果死亡的机器数量达到一半，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

- zookeeper 可伸缩扩展性原理与设计
  - 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。

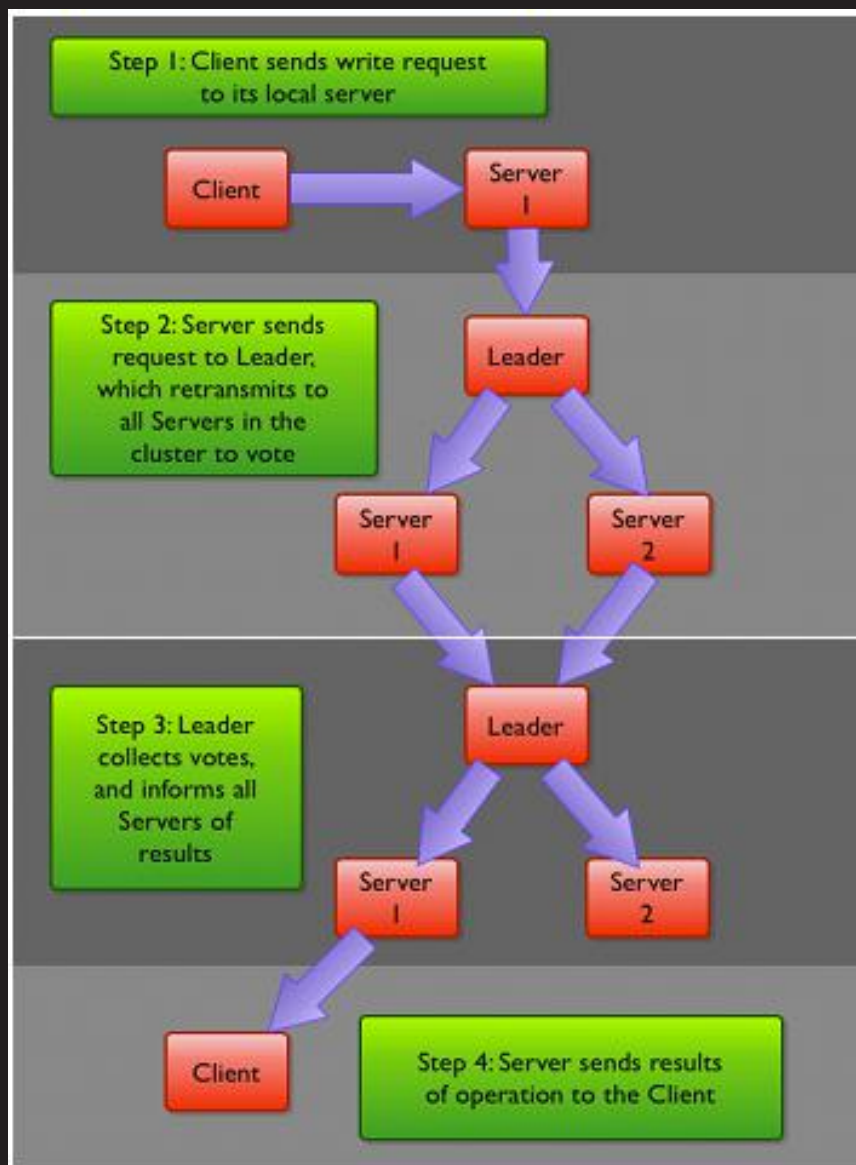


# zookeeper

- 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Proposal。然后，其他的Server对该Proposal进行Vote。之后，Leader对Vote进行收集，当Vote数量过半时Leader会向所有的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



# zookeeper

- 续上页
  - 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



# zookeeper

- 续上页

- 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



# zookeeper

- 续上页
  - 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，我们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



# zookeeper

- 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、启动服务
  - 6、查看状态





# zookeeper

- 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



# zookeeper

- ZK 集群的安装配置
  - zoo.cfg
  - server.1=node1:2888:3888
  - server.2=node2:2888:3888
  - server.3=node3:2888:3888
  - server.4=master:2888:3888:observer



# zookeeper

- 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



# zookeeper

- 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`
  - or
  - `{ echo 'stat';yes; }|telnet 192.168.4.10 2181`
  - Zookeeper [管理文档](#)



# zookeeper

- Zookeeper 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是使用Scala编写
  - kafka是一种消息中间件
- 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集群

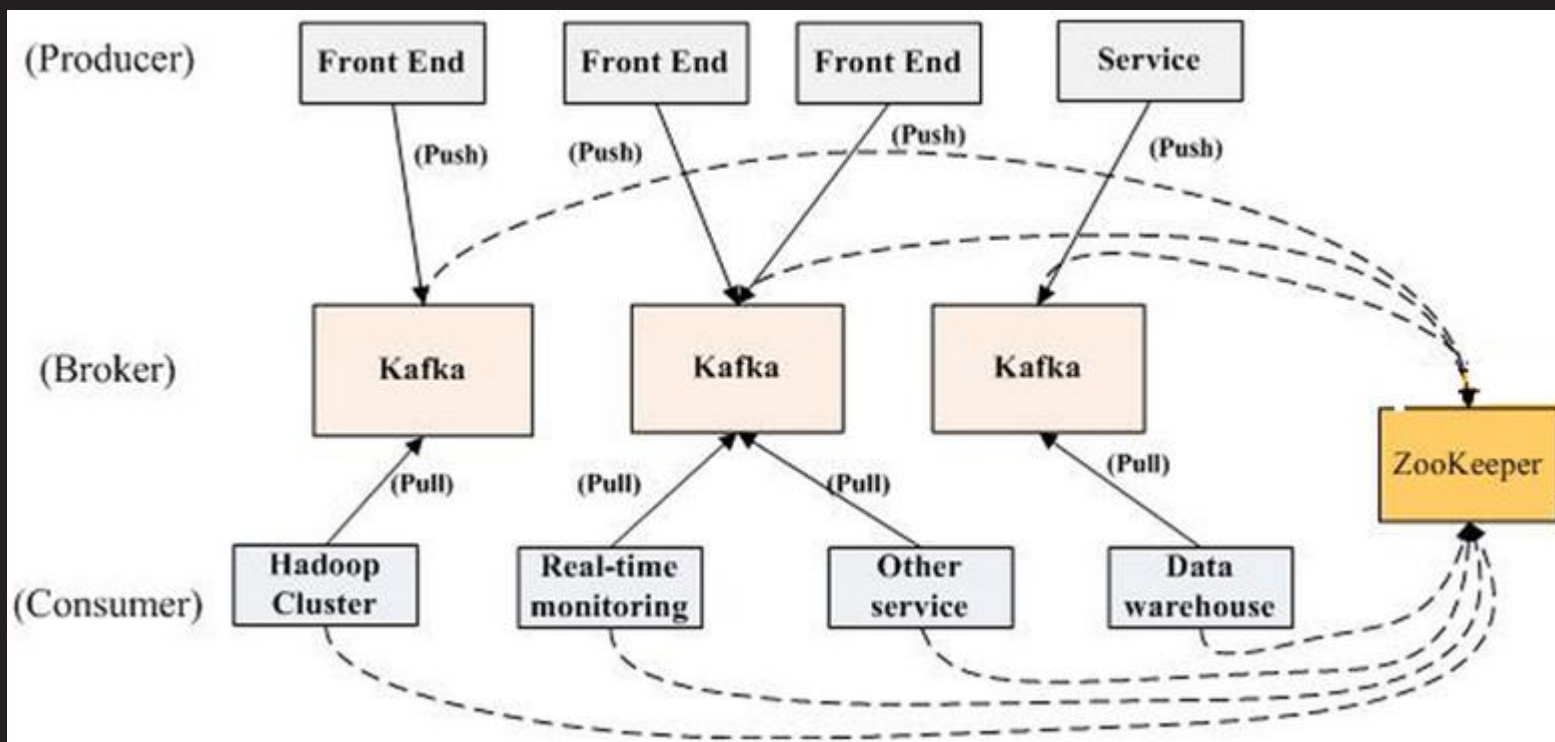
- kafka 角色与集群结构
  - producer : 生产者, 负责发布消息
  - consumer : 消费者, 负责读取处理消息
  - topic : 消息的类别
  - Parition : 每个Topic包含一个或多个Partition.
  - Broker : Kafka集群包含一个或多个服务器
  - Kafka通过Zookeeper管理集群配置, 选举leader





# kafka集群

- kafka 角色与集群结构



# kafka集群

- 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

- kafka 集群的安裝配置
- server.properties
  - broker.id
  - 每台服务器的broker.id都不能相同
  - zookeeper.connect
  - zookeeper 集群地址，不用都列出，写一部分即可



# kafka集群

- kafka 集群的安裝配置
  - 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` 命令应该能看到 kafka 模块
  - `netstat` 应该能看到 9092 在监听



# kafka集群

- 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
```



# kafka集群

- 集群验证与消息发布

- 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



# kafka集群实验

- 利用 zookeeper 搭建一个 kafka 集群
- 创建一个 topic
- 查看系统 topic 情况
- 模拟生产者发布消息
- 模拟消费者接收消息



# 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 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



# NameNode 高可用

- 为什么 NameNode 需要高可用
  - 官方提供了两种解决方案
  - HDFS with NFS
  - HDFS with QJM
  - 两种翻案异同

| NFS                  | QJM                  |
|----------------------|----------------------|
| NN                   | NN                   |
| ZK                   | ZK                   |
| ZKFailoverController | ZKFailoverController |
| NFS                  | JournalNode          |



# NameNode 高可用

- 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共享在共享存储里面，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

- 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。



# NameNode 高可用

- 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，活动状态的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 续.....
  - 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 (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 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Standby持有的namespace应该与Active保持完全同步。



# NameNode 高可用

- NameNode 高可用架构 续.....
  - NameNode 更新是很频繁的，为了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



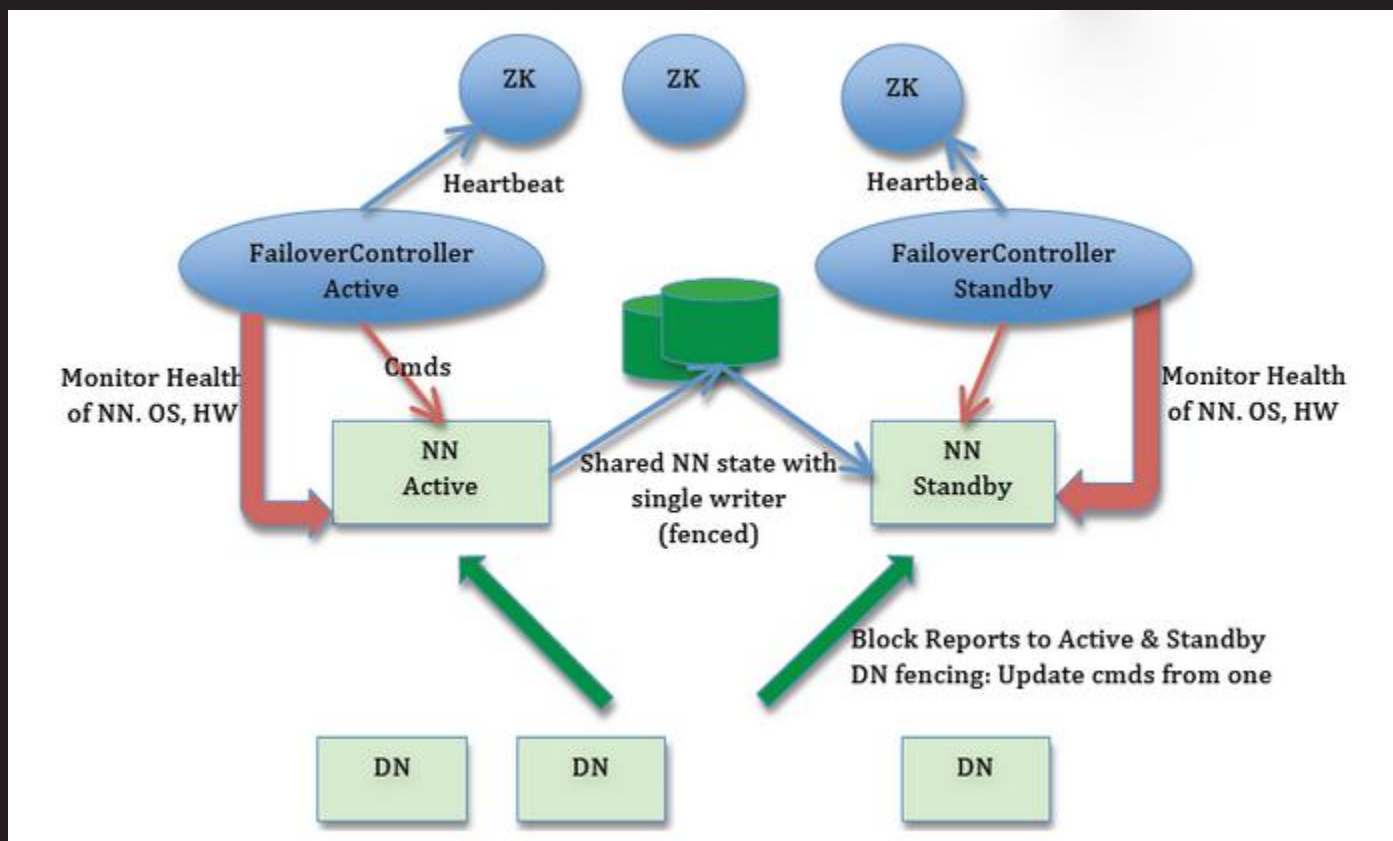
# NameNode 高可用

- NameNode 高可用架构 续.....
  - 还有一点非常重要，任何时刻，只能有一个Active NameNode，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，三节点通讯阻断，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，这中机制阻止了其他NameNode基于处于Active状态的问题。



# NameNode 高可用

- NameNode 高可用架构 续.....



# NameNode 高可用

- 系统规划

| 主机                    | 角色                    | 软件                |
|-----------------------|-----------------------|-------------------|
| 192.168.4.10          | NameNode1             | Hadoop            |
| 192.168.4.20          | NameNode2             | Hadoop            |
| 192.168.4.11<br>node1 | DataNode<br>Zookeeper | HDFS<br>Zookeeper |
| 192.168.4.12<br>node2 | DataNode<br>Zookeeper | HDFS<br>Zookeeper |
| 192.168.4.13<br>node3 | DataNode<br>Zookeeper | HDFS<br>Zookeeper |

# NameNode 高可用

- 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>
```



# NameNode 高可用

- hdfs-site.xml

```
<property>  
  <name>dfs.replication</name>  
  <value>1</value>  
</property>
```

- secondarynamenode 在高可用里面没有用途，这里把他关闭
- namenode 在后面定义



# NameNode 高可用

- hdfs-site.xml 续 .....
  - <!-- 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>
```



# NameNode 高可用

- hdfs-site.xml 续 .....
- 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>
```



# NameNode 高可用

- hdfs-site.xml 续 .....
- 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>
```



# NameNode 高可用

- hdfs-site.xml 续 .....
  - 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>
```





# NameNode 高可用

- hdfs-site.xml 续 .....
- 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>
```



# NameNode 高可用

- hdfs-site.xml 续 .....

- 配置隔离机制为 ssh

```
<property>
  <name>dfs.ha.fencing.methods</name>
  <value>sshfence</value>
</property>
```

- 指定秘钥的位置

```
<property>
  <name>dfs.ha.fencing.ssh.private-key-files</name>
  <value>/home/hadoop/.ssh/id_rsa</value>
</property>
```



# NameNode 高可用

- hdfs-site.xml 续 .....

- 开启自动故障转移

```
<property>  
  <name>dfs.ha.automatic-failover.enabled</name>  
  <value>true</value>  
</property>
```



# NameNode 高可用

- 同步数据与高可用验证
  - 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

- 同步数据与高可用验证
  - 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 -initializeSharedEdits`



# NameNode 高可用

- 同步数据与高可用验证
  - 停止 JournalNode  
`sbin/hadoop-daemon.sh stop journalnode`
  - 启动 dfs  
`./sbin/start-dfs.sh start`



# NameNode 高可用

- 同步数据与高可用验证

- 验证配置

- ```
bin/hadoop dfsadmin -report
```

- 查看集群状态

- ```
bin/hdfs haadmin -getServiceState nn1
```

- ```
bin/hdfs haadmin -getServiceState nn2
```



# NameNode 高可用

- 同步数据与高可用验证

- 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，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 高可用

- 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>
```



# Yarn 高可用

- yarn-site.xml 配置

```
<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>
```



# Yarn 高可用

- yarn-site.xml 配置

```
<property>
```

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

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

```
    <description>For multiple zk services, separate them  
with comma</description>
```

```
</property>
```

```
<property>
```

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

```
    <value>yarn-ha</value>
```

```
</property>
```



# Yarn 高可用

- yarn-site.xml 续 .....

```
<property>
```

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

```
<value>node1 </value>
```

```
</property>
```

```
<property>
```

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

```
<value>node2 </value>
```

```
</property>
```



# Yarn 高可用

- 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`



# 课程知识点总结与实验

- zookeeper 集群搭建
  - zookeeper leader 的选举
- kafka 集群搭建
  - 实现消息订阅发布
- Hadoop 高可用集群的原理与搭建方式
  - namenode & yarn 高可用
  - 上传分析文件，故障转移测试





# 总结答疑

---