

hadoop 生态系统概况.....	3
Hadoop YARN 产生背景.....	3
Hadoop 1.0 和 2.0.....	4
HDFS 2.0 的 HA 及 Federation.....	4
NN HA 的两种方案.....	4
NN Federation.....	5
部署事例图.....	5
Hadoop2.0 YARN 介绍.....	6
ResourceManager.....	6
NodeManager.....	6
ApplicationMaster.....	6
Container.....	7
Hadoop2.0 单机简单测试环境搭建.....	7
linux 系统基本环境配置.....	7
hosts 文件（可配也可不配）.....	7
时间同步.....	7
关闭防火墙以及 selinux.....	7
安装 JAVA JDK 系统软件.....	7
配置环境变量.....	8
配置免密码登录环境.....	8
Hadoop 配置文件修改.....	9
修改 hadoop-env.sh 文件中的 JAVA_HOME 变量.....	9
修改 cat core-site.xml.....	9
修改 hdfs-site.xml.....	9
修改 mapred-site.xml.....	9
修改 yarn-site.xml.....	10
hadoop 启动与测试.....	10
相关命令.....	10
启动后测试.....	11
执行 Hadoop pi 运行样例检查.....	12
Hadoop 分布式环境配置.....	12
创建相关目录.....	13
创建 namenode 目录.....	13
创建 DataNode and YARN NodeManager 目录.....	13
YARN and MapReduce 的内存分配计算.....	13
给系统预留内存.....	14
Minimum Container Size 的建议.....	14
内存分配建议（计算方法）.....	14
常见配置参数解析（摘自董西成老师的博客）.....	15
RM 与 NM 相关参数.....	15
权限与日志聚集相关参数.....	18
MapReduce 相关参数.....	19

Fair Scheduler 相关参数.....	21
Capacity Scheduler 相关参数.....	23
Hadoop 配置文件解析样例.....	24
core-site.xml 配置文件样例.....	25
hdfs-site.xml 配置文件样例.....	26
yarn-site.xml 配置文件样例.....	30
mapred-site.xml 配置文件样例.....	33
集群启动及测试.....	36
学习方向.....	37
源码编译环境配置.....	39
需要的软件.....	39
安装所需软件.....	39
常见错误.....	40
HDFS HA and Federation.....	42
HA.....	42
High Availability With QJM.....	43
High Availability With NFS.....	53
单台 namenode 扩展 standby.....	55
Federation.....	59
Hbase 安装于配置.....	60
HBase 介绍.....	60
搭建 HBase 单机环境.....	60
搭建 Hbase 分布式环境.....	62
Hbase 原理简介.....	64
Hbase 配置文件.....	64
Hbase 数据模型.....	66
Hbase 命令.....	66
Storm 原理简介.....	67
Storm 安装配置.....	70
Flume 分布式日志采集系统.....	72
原理介绍.....	72
架构理解.....	73
部署配置.....	77
开发相关.....	82

hadoop 生态系统概况

The Apache™ Hadoop® project develops open-source software for reliable, scalable, distributed computing. The Apache Hadoop software library is a framework that allows for the distributed processing of large data sets across clusters of computers using simple programming models. It is designed to scale up from single servers to thousands of machines, each offering local computation and storage. Rather than rely on hardware to deliver high-availability, the library itself is designed to detect and handle failures at the application layer, so delivering a highly-available service on top of a cluster of computers, each of which may be prone to failures. The project includes these modules:

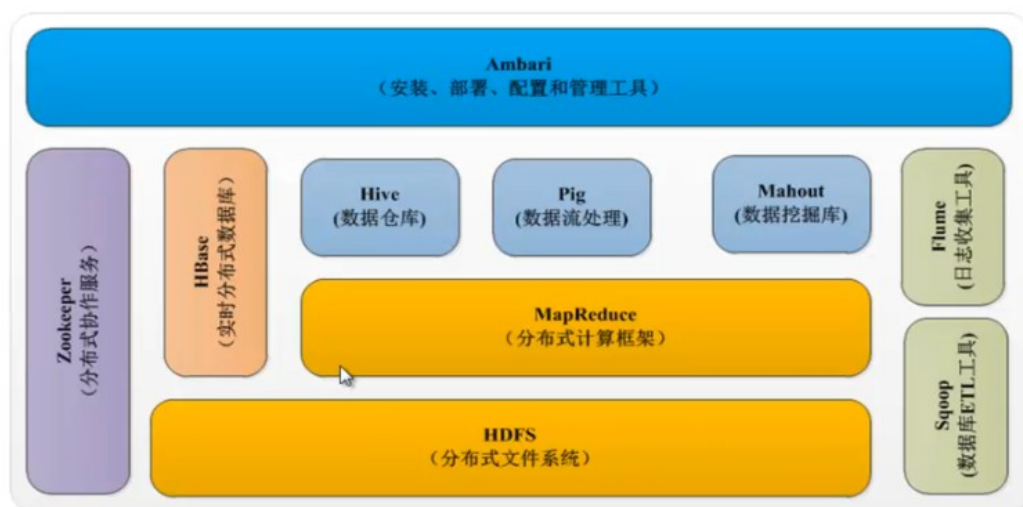
Hadoop Common: The common utilities that support the other Hadoop modules.

Hadoop Distributed File System (HDFS™): A distributed file system that provides high-throughput access to application data.

Hadoop YARN: A framework for job scheduling and cluster resource management.

Hadoop MapReduce: A YARN-based system for parallel processing of large data sets.

参考：<http://hadoop.apache.org/>



Hadoop YARN 产生背景

直接源于 MRv1 在几个方面的缺陷

❑ 扩展性受限

❑ 单点故障

❑ 难以支持 MR 之外的计算

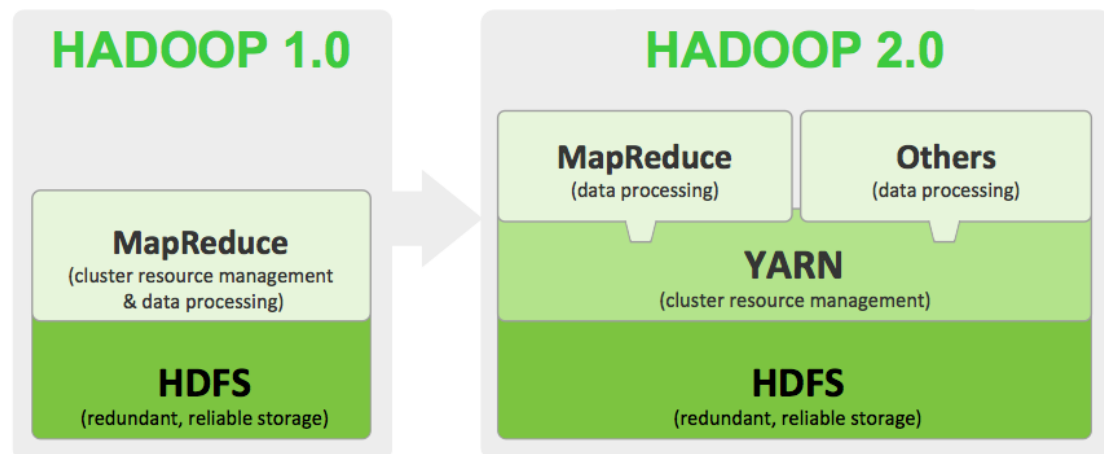
多计算框架各自为战，数据共享困难

❓MR：离线计算框架

❓Storm：实时计算框架

❓Spark：内存计算框架

Hadoop 1.0 和 2.0



❓●Hadoop 2.0 由 HDFS、MapReduce 和 YARN 三个分支构成；

❓●HDFS：NN Federation、HA；

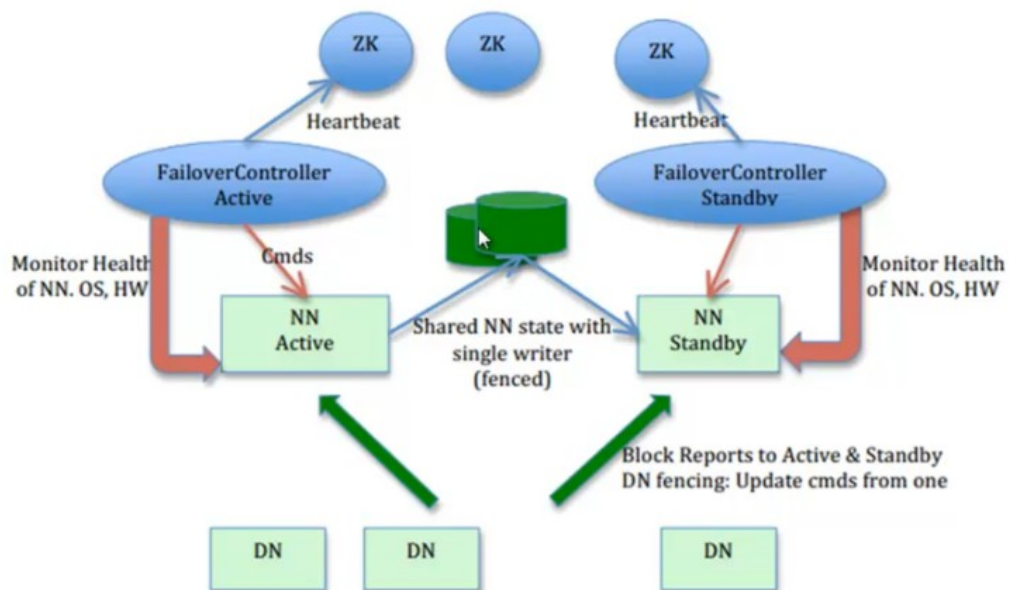
❓●MapReduce：运行在 YARN 上的 MR；

❓●YARN：资源管理系统

HDFS 2.0 的 HA 及 Federation

NN HA 的两种方案

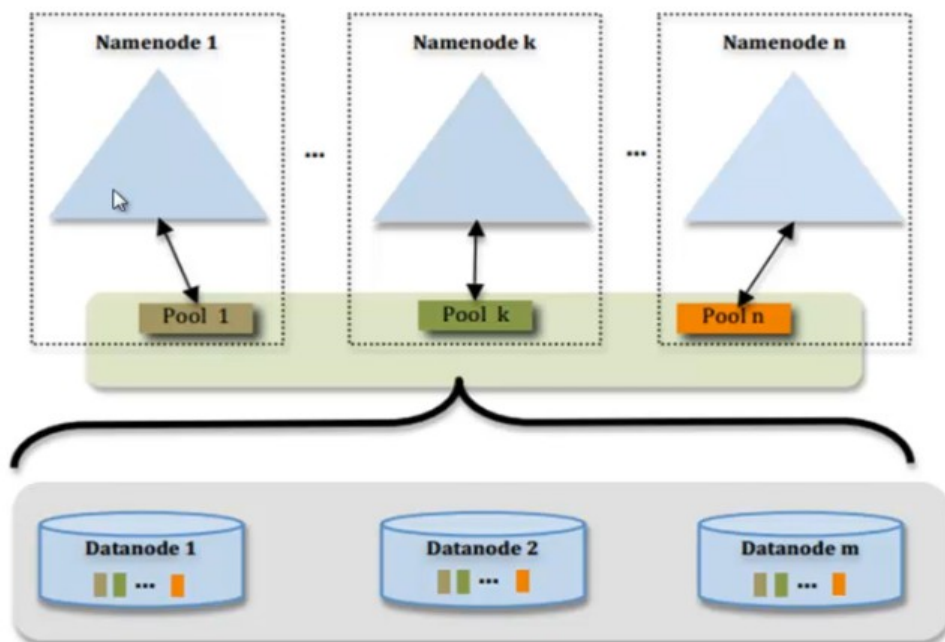
- 基于 NFS 共享存储解决方案



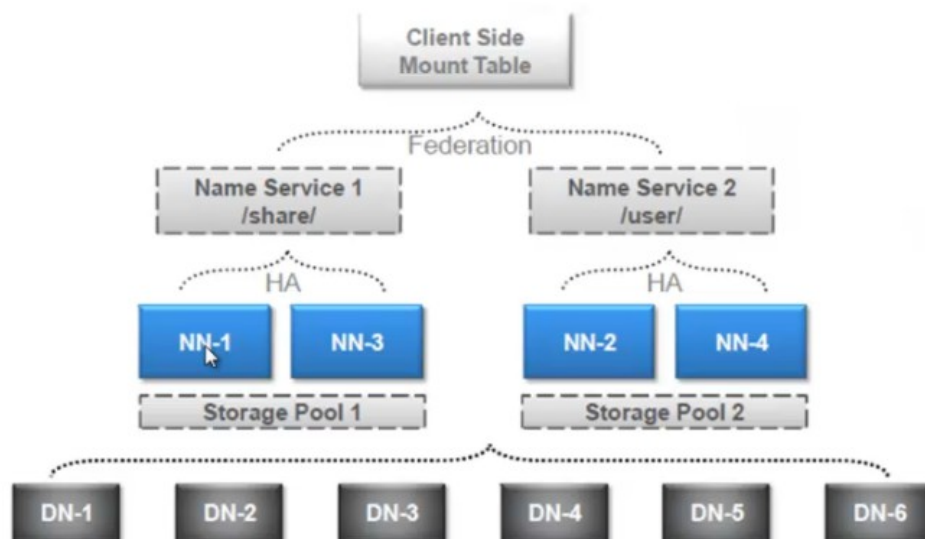
- 基于 QJM 解决方案

NN Federation

- 多个 NN，每个分管一部分目录
- NN 共用 DN

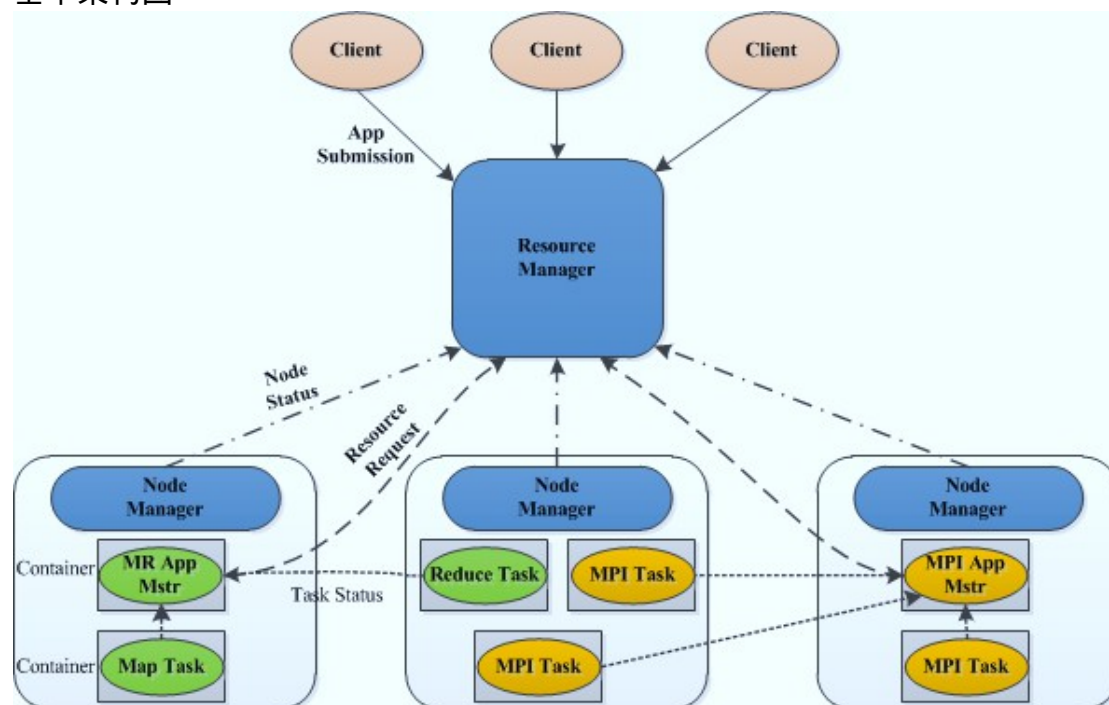


部署事例图



Hadoop2.0 YARN 介绍

基本架构图



YARN 各个模块

ResourceManager

- ❓处理客户端请求
- ❓启动/监控 ApplicationMaster
- ❓监控 NodeManager
- ❓资源分配与调度

NodeManager

- ❓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

ApplicationMaster

- ❓数据切分
- ❓为应用程序申请资源，并分配给内部任务
- ❓任务监控与容错

Container

- ❓对任务运行环境的抽象，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

Hadoop2.0 单机简单测试环境搭建

单机测试先把 hadoop 运行起来，至于配置文件都用默认，后续在考虑每个配置的意义，hadoop 2.0 对内存要求较高，越大越好。

linux 系统基本环境配置

hosts 文件（可配也可不配）

211.211.211.210 master

这样在后面的配置文件中可以用域名代替 ip 地址

时间同步

如果是多台机器需要配置时间同步，crontab -e 增加如下行

10 */2 * * * /usr/sbin/ntpdate 114.80.81.1;hwclock -w

手动直接执行/usr/sbin/ntpdate 114.80.81.1;hwclock -w 就行

关闭防火墙以及 selinux

如果不关闭会导致 master 无法访问 node 节点，当然如果你能配置 iptables 策略也是可以的，selinux 一般生产环境都是关闭的

安装 JAVA JDK 系统软件

下载

wget http://www.java.net/download/jdk6/6u38/promoted/b04/binaries/jdk-6u38-ea-bin-b04-linux-amd64-31_oct_2012-rpm.bin

其他 jdk 版本下载地址

<https://jdk6.java.net/download.html>

安装

chmod 755 jdk-6u38-ea-bin-b04-linux-amd64-31_oct_2012-rpm.bin
./jdk-6u38-ea-bin-b04-linux-amd64-31_oct_2012-rpm.bin

配置环境变量

vi /etc/profile.d/java.sh

export JAVA_HOME=/usr/java/jdk1.6.0_38/

export HADOOP_HOME=/homehadoop/hadoop-2.2.0/


```
export PATH=$JAVA_HOME/bin:$HADOOP_HOME/bin: $HADOOP_HOME/sbin:$PATH
```

#手动立即生效

```
source /etc/profile
```

配置免密码登录环境

却换到 hadoop 用户

```
su hadoop
```

```
cd
```

创建密钥并生成自动应答文件

```
mkdir .ssh
```

```
ssh-keygen -q -t rsa -N "" -f /home/hadoop/.ssh/id_rsa 或者是直接执行 ssh-keygen 一路回车
```

```
cd .ssh
```

```
[hadoop@master .ssh]$ ssh-copy-id
Usage: /usr/bin/ssh-copy-id [-i [identity_file]] [user@]machine
[hadoop@master .ssh]$ ssh-copy-id localhost
hadoop@localhost's password:
Now try logging into the machine, with "ssh 'localhost'", and check in:

    .ssh/authorized_keys

to make sure we haven't added extra keys that you weren't expecting.

[hadoop@master .ssh]$ ls
authorized_keys  id_rsa  id_rsa.pub  known_hosts
```

如果是其他机器把 localhost 改为其他机器的 ip 就行，当然也可以采用另一种方法直接拷贝公钥 id_rsa.pub 内容到其他机器的 .ssh/authorized_keys 文件中，但需要注意权限 644

比如集群环境把 id_rsa.pub 复制到 node1:/home/hadoop/.ssh/authorized_keys，权限改为 644，就可以免密码登录了

Hadoop 配置文件修改

修改 hadoop-env.sh 文件中的 JAVA_HOME 变量

```
export JAVA_HOME=/usr/java/jdk1.6.0_38/
```

不知道为什么默认配置的 \$JAVA_HOME 无法从系统变量中获取，如果这里没有修改，会导致启动 nodemanager 是报错如下错误

```
Error: JAVA_HOME is not set and could not be found.
```

修改 cat core-site.xml

```
<?xml version="1.0" encoding="UTF-8"?>  
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>  
<configuration>  
  <property>  
    <name>fs.default.name</name>  
    <value>hdfs://master:8020</value>  
  </property>  
</configuration>
```

修改 hdfs-site.xml

```
<?xml version="1.0" encoding="UTF-8"?>  
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>  
<configuration>  
  <property>  
    <name>dfs.replication</name>  
    <value>1</value>  
  </property>  
</configuration>
```

修改 mapred-site.xml

```
<?xml version="1.0"?>  
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>  
<configuration>  
  <property>  
    <name>mapreduce.framework.name</name>  
    <value>yarn</value>  
  </property>  
</configuration>
```

修改 yarn-site.xml

```
<?xml version="1.0"?>  
<configuration>
```

```
<property>
  <name>yarn.nodemanager.aux-services</name>
  <value>mapreduce_shuffle</value>
</property>
</configuration>
```

hadoop 启动与测试

相关命令

启动之前 HDFS format

```
hdfs namenode -format
```

启动 namenode、datanode

```
hadoop-daemon.sh start/stop namenode
```

```
hadoop-daemon.sh start/stop datanode
```

或者是

```
start-dfs.sh
```

 这个命令同时启动 namenode、datanode，还有不知道为什么还会启动 secondarynamenode，hadoop2.0 中已经有 HA 及 Federation 方案代替了

启动 RM、NM

```
yarn-daemon.sh start/stop resourcemanager
```

```
yarn-daemon.sh start/stop nodemanager
```

或者

```
start-yarn.sh
```

测试是否启动成功 jps

```
[hadoop@master mapreduce]$ jps
15572 Jps
1460 NameNode
1901 ResourceManager
1547 DataNode
1996 NodeManager
```

如果少了哪个没有启动，可以查看日志，看看是什么原因

```
tail -100f $HADOOP_HOME/logs/hadoop-hadoop-namenode-master.log
```

hdfs 文件拷贝

```
[hadoop@master mapreduce]$ hadoop fs
```

```
Usage: hadoop fs [generic options]
```

```
[-cat [-ignoreCrc] <src> ...]
```

```
[-df [-h] [<path> ...]]
```

```
[-get [-p] [-ignoreCrc] [-crc] <src> ... <localdst>]
```

```
[-ls [-d] [-h] [-R] [<path> ...]]
```

```
[-mkdir [-p] <path> ...]
```

```
[-mv <src> ... <dst>]
```

```
[-put [-f] [-p] <localsrc> ... <dst>]
```

```
[-rm [-f] [-r] [-R] [-skipTrash] <src> ...]
```

```
[-rmdir [--ignore-fail-on-non-empty] <dir> ...]
```

这些是比较常用的

比如：

创建目录：hadoop fs -mkdir /app

上传文件：hadoop fs -put hadoop-2.2.0/etc/hadoop/* /app

启动后测试

访问页面 master:8088

Cluster Metrics

Apps Submitted	Apps Pending	Apps Running	Apps Completed	Containers Running	Memory Used	Memory Total	Memory Reserved	Active Nodes	Decommissioned Nodes	...
2	0	0	2	0	0 B	8 GB	0 B	1	0	...

Showing 0 to 0 of 0 entries (filtered from 2 total entries)

访问页面 master : 50070

NameNode 'master:8020' (active)

Property	Value
Started:	Sat Mar 08 14:59:08 CST 2014
Version:	2.2.0, 1529768
Compiled:	2013-10-07T06:28Z by hortonmu from branch-2.2.0
Cluster ID:	CID-8f96e976-a3ca-4583-9f06-5e7f132bd491
Block Pool ID:	BP-1443406248-211.211.211.210-1394260982571

Cluster Summary

Security is OFF

44 files and directories, 32 blocks = 76 total.

Heap Memory used 17.41 MB is 45% of Committed Heap Memory 38.35 MB. Max Heap Memory is 966.69 MB.

Non Heap Memory used 31.53 MB is 95% of Committed Non Heap Memory 32.88 MB. Max Non Heap Memory is 130 MB.

Property	Value
Configured Capacity	19.69 GB

很多配置都是可以自定义的，想上面的端口、内存使用分配、heapsize、数据存放目录（默认都存储在/tmp 目录下）等

执行 Hadoop pi 运行样例检查

```
[hadoop@master mapreduce]$ hadoop jar hadoop-mapreduce-examples-2.2.0.jar pi 10 100
Number of Maps = 10
Samples per Map = 100
Wrote input for Map #0
.....
Wrote input for Map #6
14/03/08 16:09:27 INFO mapreduce.Job: map 0% reduce 0%
14/03/08 16:10:22 INFO mapreduce.Job: map 40% reduce 0%
14/03/08 16:10:23 INFO mapreduce.Job: map 60% reduce 0%
14/03/08 16:11:06 INFO mapreduce.Job: map 100% reduce 0%
  Job Counters
    Launched map tasks=10
    Launched reduce tasks=1
  Map-Reduce Framework
    Map input records=10
    Map output records=20
.....
  File Output Format Counters
    Bytes Written=97
Job Finished in 116.936 seconds
Estimated value of Pi is 3.14800000000000000000
OK!这里就运行成功了，单机环境搭建成功
```

Hadoop 分布式环境配置

linux 的基本环境配置参考前面，这里创建一个自定义的集群，很多默认选项并不适合生产环境。

这里实验配置有三台机器：

```
nn namenode 211.211.211.210
rm resourcemanager 211.211.211.211
dn datanode 211.211.211.212
```

nodemanager 和 datanode 在三台机器上都启动

创建相关目录

创建 namenode 目录

只需要在运行 namenode 的节点的机器上执行：

```
mkdir -p $DFS_NAME_DIR;  
chown -R $HADOOP:$HADOOP_GROUP $DFS_NAME_DIR;  
chmod -R 755 $DFS_NAME_DIR;
```

\$DFS_NAME_DIR 存储 namenode 元数据目录/home/dir/dfs/name
\$ HADOOP:\$HADOOP_GROUP 用户和组自定义 hadoop:hadoop

创建 DataNode and YARN NodeManager 目录

在所有的 datanode 上执行以下命令

```
mkdir -p $DFS_DATA_DIR;  
chown -R $ HADOOP:$HADOOP_GROUP $DFS_DATA_DIR;  
chmod -R 750 $DFS_DATA_DIR;
```

\$DFS_DATA_DIR 存储集群数据的目录/home/dir/dfs/data

在所有的 datanode and resourcemanager 上执行一下命令

```
mkdir -p $YARN_LOCAL_DIR;  
chown -R $ HADOOP:$HADOOP_GROUP $YARN_LOCAL_DIR;  
chmod -R 755 $YARN_LOCAL_DIR;
```

\$YARN_LOCAL_DIR 存储 YARN 的临时数据目录/home/dir/yarn/local

YARN and MapReduce 的内存分配计算

在 YARN 中，资源调度器（ResourceScheduler）是一个非常核心的部件，它负责将各个节点上的资源封装成 container，并按照一定的约束条件（按队列分配，每个队列有一定的资源分配上限等）分配给各个 application。

当前 YARN 支持内存和 CPU 两种资源类型的管理和分配。当 NodeManager 启动时，会向 ResourceManager 注册，而注册信息中会包含该节点可分配的

CPU 和内存总量，这两个值均可通过配置选项设置，对于集群中不通配置的机器只需要修改这些资源分配参数就行了。

下面来了解一下这些参数怎么配置，主要在两个配置文件中，core-site.xml 和 yarn-site.xml

hortonworks.com 网站给出了建议的计算方法：

给系统预留内存

Total Memory per Node	Recommended System Memory	Reserved HBase Memory
4 GB	1 GB	1 GB
8 GB	2 GB	1 GB
16 GB	2 GB	2 GB
24 GB	4 GB	4 GB
48 GB	6 GB	8 GB
64 GB	8 GB	8 GB
72 GB	8 GB	8 GB
96 GB	12 GB	16 GB
128 GB	24 GB	24 GB

Minimum Container Size 的建议

Total RAM per Node Recommended Minimum Container Size	Total RAM per Node Recommended Minimum Container Size
Less than 4 GB 256 MB	Less than 4 GB 256 MB
Between 4 GB and 8 GB 512 MB	Between 4 GB and 8 GB 512 MB
Between 8 GB and 24 GB 1024 MB	Between 8 GB and 24 GB 1024 MB
Above 24 GB 2048 MB	Above 24 GB 2048 MB

内存分配建议（计算方法）

计算公式：

Containers = minimum of (2*CORES, 1.8*DISKS, (Total available RAM) /MIN_CONTAINER_SIZE)

RAM-per-Container = maximum of (MIN_CONTAINER_SIZE, (Total Available RAM) /Containers))

举例：

Cluster nodes have 12 CPU cores, 48 GB RAM, and 12 disks.

Reserved Memory = 6 GB reserved for system memory

Min Container size = 2 GB

参数含义可以参见下面提供的网址

常见配置参数解析（摘自董西成老师的博客）

RM 与 NM 相关参数

注意，配置这些参数前，应充分理解这几个参数的含义，以防止误配给集群带来的隐患。另外，这些参数均需要在 yarn-site.xml 中配置。

1. ResourceManager 相关配置参数

（1）yarn.resourcemanager.address

参数解释：ResourceManager 对客户端暴露的地址。客户端通过该地址向 RM 提交应用程序，杀死应用程序等。

默认值：\${yarn.resourcemanager.hostname}:8032

(2) yarn.resourcemanager.scheduler.address

参数解释：ResourceManager 对 ApplicationMaster 暴露的访问地址。ApplicationMaster 通过该地址向 RM 申请资源、释放资源等。

默认值：\${yarn.resourcemanager.hostname}:8030

(3) yarn.resourcemanager.resource-tracker.address

参数解释：ResourceManager 对 NodeManager 暴露的地址。NodeManager 通过该地址向 RM 汇报心跳，领取任务等。

默认值：\${yarn.resourcemanager.hostname}:8031

(4) yarn.resourcemanager.admin.address

参数解释：ResourceManager 对管理员暴露的访问地址。管理员通过该地址向 RM 发送管理命令等。

默认值：\${yarn.resourcemanager.hostname}:8033

(5) yarn.resourcemanager.webapp.address

参数解释：ResourceManager 对外 web ui 地址。用户可通过该地址在浏览器中查看集群各类信息。

默认值：\${yarn.resourcemanager.hostname}:8088

(6) yarn.resourcemanager.scheduler.class

参数解释：启用的资源调度器主类。目前可用的有 FIFO、Capacity Scheduler 和 Fair Scheduler。

默认值：

org.apache.hadoop.yarn.server.resourcemanager.scheduler.capacity.CapacityScheduler

(7) yarn.resourcemanager.resource-tracker.client.thread-count

参数解释：处理来自 NodeManager 的 RPC 请求的 Handler 数目。

默认值：50

(8) yarn.resourcemanager.scheduler.client.thread-count

参数解释：处理来自 ApplicationMaster 的 RPC 请求的 Handler 数目。

默认值：50

(9) yarn.scheduler.minimum-allocation-mb / yarn.scheduler.maximum-allocation-mb

参数解释：单个可申请的最小/最大内存资源量。比如设置为 1024 和 3072，则运行 MapReduce 作业时，每个 Task 最少可申请 1024MB 内存，最多可申请 3072MB 内存。

默认值：1024/8192

(10) yarn.scheduler.minimum-allocation-vcores / yarn.scheduler.maximum-

allocation-vcores

参数解释：单个可申请的最小/最大虚拟 CPU 个数。比如设置为 1 和 4，则运行 MapRedce 作业时，每个 Task 最少可申请 1 个虚拟 CPU，最多可申请 4 个虚拟 CPU。什么是虚拟 CPU，可阅读我的这篇文章：“YARN 资源调度器剖析”。

默认值：1/32

(11) yarn.resourcemanager.nodes.include-path
/yarn.resourcemanager.nodes.exclude-path

参数解释：NodeManager 黑白名单。如果发现若干个 NodeManager 存在问题，比如故障率很高，任务运行失败率高，则可以将之加入黑名单中。注意，这两个配置参数可以动态生效。（调用一个 refresh 命令即可）

默认值：“”

(12) yarn.resourcemanager.nodemanager.heartbeat-interval-ms

参数解释：NodeManager 心跳间隔

默认值：1000（毫秒）

2. NodeManager 相关配置参数

(1) yarn.nodemanager.resource.memory-mb

参数解释：NodeManager 总的可用物理内存。注意，该参数是不可修改的，一旦设置，整个运行过程中不可动态修改。另外，该参数的默认值是 8192MB，即使你的机器内存不够 8192MB，YARN 也会按照这些内存来使用（傻不傻？），因此，这个值通过一定要配置。不过，Apache 已经正在尝试将该参数做成可动态修改的。

默认值：8192

(2) yarn.nodemanager.vmem-pmem-ratio

参数解释：每使用 1MB 物理内存，最多可用的虚拟内存数。

默认值：2.1

(3) yarn.nodemanager.resource.cpu-vcores

参数解释：NodeManager 总的可用虚拟 CPU 个数。

默认值：8

(4) yarn.nodemanager.local-dirs

参数解释：中间结果存放位置，类似于 1.0 中的 mapred.local.dir。注意，这个参数通常会配置多个目录，已分摊磁盘 IO 负载。

默认值：\${hadoop.tmp.dir}/nm-local-dir

(5) yarn.nodemanager.log-dirs

参数解释：日志存放地址（可配置多个目录）。

默认值：\${yarn.log.dir}/userlogs

(6) yarn.nodemanager.log.retain-seconds

参数解释：NodeManager 上日志最多存放时间（不启用日志聚集功能时有效）。

默认值：10800（3 小时）

（7）yarn.nodemanager.aux-services

参数解释：NodeManager 上运行的附属服务。需配置成 mapreduce_shuffle，才可运行 MapReduce 程序

<http://dongxicheng.org/mapreduce-nextgen/hadoop-yarn-configurations-resource-manager-nodemanager/>

权限与日志聚集相关参数

注意，配置这些参数前，应充分理解这几个参数的含义，以防止误配给集群带来的隐患。另外，这些参数均需要在 yarn-site.xml 中配置。

1. 权限相关配置参数

这里的权限由三部分组成，分别是：（1）管理员和普通用户如何区分（2）服务级别的权限，比如哪些用户可以向集群提交 ResourceManager 提交应用程序，（3）队列级别的权限，比如哪些用户可以向队列 A 提交作业等。

管理员列表由参数 yarn.admin.acl 指定。

服务级别的权限是通过配置 hadoop-policy.xml 实现的，这个与 Hadoop 1.0 类似队列级别的权限是由对应的资源调度器内部配置的，比如 Fair Scheduler 或者 Capacity Scheduler 等，具体阅读后面几节。

2. 日志聚集相关配置参数

日志聚集是 YARN 提供的日志中央化管理功能，它可将运行完成的 Container/任务日志上传到 HDFS 上，从而减轻 NodeManager 负载，且提供一个中央化存储和分析机制。默认情况下，Container/任务日志存在在各个 NodeManager 上，如果启用日志聚集功能需要额外的配置。

（1）yarn.log-aggregation-enable

参数解释：是否启用日志聚集功能。

默认值：false

（2）yarn.log-aggregation.retain-seconds

参数解释：在 HDFS 上聚集的日志最多保存多长时间。

默认值：-1

（3）yarn.log-aggregation.retain-check-interval-seconds

参数解释：多长时间检查一次日志，并将满足条件的删除，如果是 0 或者负数，则为上一个值的 1/10。

默认值：-1

（4）yarn.nodemanager.remote-app-log-dir

参数解释：当应用程序运行结束后，日志被转移到的 HDFS 目录（启用日志聚集功能时有效）。

默认值：/tmp/logs

(5) yarn.log-aggregation.retain-seconds

参数解释：远程日志目录子目录名称（启用日志聚集功能时有效）。

默认值：日志将被转移到目录

`${yarn.nodemanager.remote-app-log-dir}/${user}/${thisParam}`下

网址：

<http://dongxicheng.org/mapreduce-nextgen/hadoop-yarn-configurations-log-aggregation/>

MapReduce 相关参数

MapReduce 相关配置参数分为两部分，分别是 JobHistory Server 和应用程序参数，Job History 可运行在一个独立节点上，而应用程序参数则可存放在 `mapred-site.xml` 中作为默认参数，也可以在提交应用程序时单独指定，注意，如果用户指定了参数，将覆盖掉默认参数。

以下这些参数全部在 `mapred-site.xml` 中设置。

1. MapReduce JobHistory 相关配置参数

在 JobHistory 所在节点的 `mapred-site.xml` 中配置。

(1) **mapreduce.jobhistory.address**

参数解释：MapReduce JobHistory Server 地址。

默认值：0.0.0.0:10020

(2) **mapreduce.jobhistory.webapp.address**

参数解释：MapReduce JobHistory Server Web UI 地址。

默认值：0.0.0.0:19888

(3) **mapreduce.jobhistory.intermediate-done-dir**

参数解释：MapReduce 作业产生的日志存放位置。

默认值：/mr-history/tmp

(4) **mapreduce.jobhistory.done-dir**

参数解释：MR JobHistory Server 管理的日志的存放位置。

默认值：/mr-history/done

2. MapReduce 作业配置参数

可在客户端的 `mapred-site.xml` 中配置，作为 MapReduce 作业的缺省配置参数。也可以在作业提交时，个性化指定这些参数。

参数名称	缺省值	说明
<code>mapreduce.job.name</code>		作业名称
<code>mapreduce.job.priority</code>	NORMAL	作业优先级
<code>yarn.app.mapreduce.am.resource.mb</code>	1536	MR ApplicationMaster 占用的内存量
<code>yarn.app.mapreduce.am.resource.c</code>	1	MR

pu-vcores		ApplicationMaster 占用的虚拟 CPU 个数
mapreduce.am.max-attempts	2	MR ApplicationMaster 最大失败尝试次数
mapreduce.map.memory.mb	1024	每个 Map Task 需要的内存量
mapreduce.map.cpu.vcores	1	每个 Map Task 需要的虚拟 CPU 个数
mapreduce.map.maxattempts	4	Map Task 最大失败尝试次数
mapreduce.reduce.memory.mb	1024	每个 Reduce Task 需要的内存量
mapreduce.reduce.cpu.vcores	1	每个 Reduce Task 需要的虚拟 CPU 个数
mapreduce.reduce.maxattempts	4	Reduce Task 最大失败尝试次数
mapreduce.map.speculative	false	是否对 Map Task 启用推测执行机制
mapreduce.reduce.speculative	false	是否对 Reduce Task 启用推测执行机制
mapreduce.job.queueName	default	作业提交到的队列
mapreduce.task.io.sort.mb	100	任务内部排序缓冲区大小
mapreduce.map.sort.spill.percent	0.8	Map 阶段溢写文件的阈值（排序缓冲区大小的百分比）
mapreduce.reduce.shuffle.parallelcopies	5	Reduce Task 启动的并发拷贝数据的线程数目

注意，MRv2 重新命名了 MRv1 中的所有配置参数，但兼容 MRv1 中的旧参数，只不过会打印一条警告日志提示用户参数过期。MapReduce 新旧参数对照表可参考 Java 类 `org.apache.hadoop.mapreduce.util.ConfigUtil`，举例如下：

过期参数名	新参数名
-------	------

mapred.job.name	mapreduce.job.name
mapred.job.priority	mapreduce.job.priority
mapred.job.queue.name	mapreduce.job.queue.name
mapred.map.tasks.speculative.execution	mapreduce.map.speculative
mapred.reduce.tasks.speculative.execution	mapreduce.reduce.speculative
io.sort.factor	mapreduce.task.io.sort.factor
io.sort.mb	mapreduce.task.io.sort.mb

网址：

<http://dongxicheng.org/mapreduce-nextgen/hadoop-yarn-configurations-mapreduce/>

Fair Scheduler 相关参数

首先在 yarn-site.xml 中，将配置参数 yarn.resourcemanager.scheduler.class 设置为 org.apache.hadoop.yarn.server.resourcemanager.scheduler.fair.FairScheduler。

[Fair Scheduler](#) 的配置选项包括两部分，其中一部分在 yarn-site.xml 中，主要用于配置调度器级别的参数，另外一部分在一个自定义配置文件（默认是 fair-scheduler.xml）中，主要用于配置各个队列的资源量、权重等信息。

想要了解 Fair Scheduler 是什么，可阅读我的这篇文章[“Hadoop 公平调度器分析”](#)。

1. 配置文件 yarn-site.xml

（1）**yarn.scheduler.fair.allocation.file**：自定义 XML 配置文件所在位置，该文件主要用于描述各个队列的属性，比如资源量、权重等，具体配置格式将在后面介绍。

（2）**yarn.scheduler.fair.user-as-default-queue**：当应用程序未指定队列名时，是否指定用户名作为应用程序所在的队列名。如果设置为 false 或者未设置，所有未知队列的应用程序将被提交到 default 队列中，默认值为 true。

（3）**yarn.scheduler.fair.preemption**：是否启用抢占机制，默认值是 false。

（4）**yarn.scheduler.fair.sizebasedweight**：在一个队列内部分配资源时，默认情况下，采用公平轮询的方法将资源分配给各个应用程序，而该参数则提供了另外一种资源分配方式：按照应用程序资源需求数目分配资源，即需求资源数量越多，分配的资源越多。默认情况下，该参数值为 false。

（5）**yarn.scheduler.assignmultiple**：是否启动批量分配功能。当一个节点出现大量资源时，可以一次分配完成，也可以多次分配完成。默认情况下，该参数值为 false。

（6）**yarn.scheduler.fair.max.assign**：如果开启批量分配功能，可指定一次分配

的 container 数目。默认情况下，该参数值为-1，表示不限制。

(7) **yarn.scheduler.fair.locality.threshold.node**：当应用程序请求某个节点上资源时，它可以接受的可跳过的最大资源调度机会。当按照分配策略，可将一个节点上的资源分配给某个应用程序时，如果该节点不是应用程序期望的节点，可选择跳过该分配机会暂时将资源分配给其他应用程序，直到出现满足该应用程序需的节点资源出现。通常而言，一次心跳代表一次调度机会，而该参数则表示跳过调度机会占节点总数的比例，默认情况下，该值为-1.0，表示不跳过任何调度机会。

(8) **yarn.scheduler.fair.locality.threshold.rack**：当应用程序请求某个机架上资源时，它可以接受的可跳过的最大资源调度机会。

(9) **yarn.scheduler.increment-allocation-mb**：内存规整化单位，默认是 1024，这意味着，如果一个 Container 请求资源是 1.5GB，则将被调度器规整化为 $\text{ceiling}(1.5 \text{ GB} / 1\text{GB}) * 1\text{G} = 2\text{GB}$ 。

(10) **yarn.scheduler.increment-allocation-vcores**：虚拟 CPU 规整化单位，默认是 1，含义与内存规整化单位类似。

2. 自定义配置文件

Fair Scheduler 允许用户将队列信息专门放到一个配置文件（默认是 fair-scheduler.xml），对于每个队列，管理员可配置以下几个选项：

(1) **minResources**：最少资源保证量，设置格式为“X mb, Y vcores”，当一个队列的最少资源保证量未满足时，它将优先于其他同级队列获得资源，对于不同的调度策略（后面会详细介绍），最少资源保证量的含义不同，对于 fair 策略，则只考虑内存资源，即如果一个队列使用的内存资源超过了它的最少资源量，则认为它已得到了满足；对于 drf 策略，则考虑主资源使用的资源量，即如果一个队列的主资源量超过它的最少资源量，则认为它已得到了满足。

(2) **maxResources**：最多可以使用的资源量，fair scheduler 会保证每个队列使用的资源量不会超过该队列的最多可使用资源量。

(3) **maxRunningApps**：最多同时运行的应用程序数目。通过限制该数目，可防止超量 Map Task 同时运行时产生的中间输出结果撑爆磁盘。

(4) **minSharePreemptionTimeout**：最小共享量抢占时间。如果一个资源池在该时间内使用的资源量一直低于最小资源量，则开始抢占资源。

(5) **schedulingMode/schedulingPolicy**：队列采用的调度模式，可以是 fifo、fair 或者 drf。

(6) **aclSubmitApps**：可向队列中提交应用程序的 Linux 用户或用户组列表，默认情况下为“*”，表示任何用户均可以向该队列提交应用程序。需要注意的是，该属性具有继承性，即子队列的列表会继承父队列的列表。配置该属性时，用户之间或用户组之间用“,”分割，用户和用户组之间用空格分割，比如“user1, user2 group1,group2”。

(7) **aclAdministerApps**：该队列的管理员列表。一个队列的管理员可管理该队列中的资源和应用程序，比如可杀死任意应用程序。

管理员也可为单个用户添加 maxRunningJobs 属性限制其最多同时运行的应用程序数目。此外，管理员也可通过以下参数设置以上属性的默认值：

(1) **userMaxJobsDefault**：用户的 maxRunningJobs 属性的默认值。

(2) **defaultMinSharePreemptionTimeout**：队列的 minSharePreemptionTimeout 属性的默认值。

(3) **defaultPoolSchedulingMode** : 队列的 schedulingMode 属性的默认值。

(4) **fairSharePreemptionTimeout** : 公平共享量抢占时间。如果一个资源池在该时间内使用资源量一直低于公平共享量的一半，则开始抢占资源。

【实例】假设要为一个 Hadoop 集群设置三个队列 queueA、queueB 和 queueC，其中，queueB 和 queueC 为 queueA 的子队列，且规定普通用户最多可同时运行 40 个应用程序，但用户 userA 最多可同时运行 400 个应用程序，那么可在自定义配置文件中进行如下设置：

```
1 <allocations>
2   <queue name="queueA">
3     <minResources>100 mb, 100 vcores</minResources>
4     <maxResources>150 mb, 150 vcores</maxResources>
5     <maxRunningApps>200</maxRunningApps>
6     <minSharePreemptionTimeout>300</minSharePreemptionTimeout>
7     <weight>1.0</weight>
8     <queue name="queueB">
9       <minResources>30 mb, 30 vcores</minResources>
1      <maxResources>50 mb, 50 vcores</maxResources>
10    </queue>
11    <queue name="queueC">
12      <minResources>50 mb, 50 vcores</minResources>
13      <maxResources>50 mb, 50 vcores</maxResources>
14    </queue>
15  </queue>
16  <user name="userA">
17    <maxRunningApps>400</maxRunningApps>
18  </user>
19  <userMaxAppsDefault>40</userMaxAppsDefault>
20  <fairSharePreemptionTimeout>6000</fairSharePreemptionTimeout>
21 </allocations>
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
```

网站：

<http://dongxicheng.org/mapreduce-nextgen/hadoop-yarn-configurations-fair->

scheduler/

Capacity Scheduler 相关参数

[Capacity Scheduler](#) 是 YARN 中默认的资源调度器。

想要了解 Capacity Scheduler 是什么，可阅读我的这篇文章“[Hadoop Capacity Scheduler 分析](#)”。

在 [Capacity Scheduler](#) 的配置文件中，队列 queueX 的参数 Y 的配置名称为 yarn.scheduler.capacity.queueX.Y，为了简单起见，我们记为 Y，则每个队列可以配置的参数如下：

1. 资源分配相关参数

(1) **capacity**：队列的资源容量（百分比）。当系统非常繁忙时，应保证每个队列的容量得到满足，而如果每个队列应用程序较少，可将剩余资源共享给其他队列。注意，所有队列的容量之和应小于 100。

(2) **maximum-capacity**：队列的资源使用上限（百分比）。由于存在资源共享，因此一个队列使用的资源量可能超过其容量，而最多使用资源量可通过该参数限制。

m **minimum-user-limit-percent**：每个用户最低资源保障（百分比）。任何时刻，一个队列中每个用户可使用的资源量均有一定的限制。当一个队列中同时运行多个用户的应用程序时，每个用户的使用资源量在一个最小值和最大值之间浮动，其中，最小值取决于正在运行的应用程序数目，而最大值则由 **minimum-user-limit-percent** 决定。比如，假设 **minimum-user-limit-percent** 为 25。当两个用户向该队列提交应用程序时，每个用户可使用资源量不能超过 50%，如果三个用户提交应用程序，则每个用户可使用资源量不能超过 33%，如果四个或者更多用户提交应用程序，则每个用户可用资源量不能超过 25%。

(3) **user-limit-factor**：每个用户最多可使用的资源量（百分比）。比如，假设该值为 30，则任何时刻，每个用户使用的资源量不能超过该队列容量的 30%。

2. 限制应用程序数目相关参数

(1) **maximum-applications**：集群或者队列中同时处于等待和运行状态的应用程序数目上限，这是一个强限制，一旦集群中应用程序数目超过该上限，后续提交的应用程序将被拒绝，默认值为 10000。所有队列的数目上限可通过参数 yarn.scheduler.capacity.maximum-applications 设置（可看做默认值），而单个队列可通过参数 yarn.scheduler.capacity.<queue-path>.maximum-applications 设置适合自己的值。

(2) **maximum-am-resource-percent**：集群中用于运行应用程序 ApplicationMaster 的资源比例上限，该参数通常用于限制处于活动状态的应用程序数目。该参数类型为浮点型，默认是 0.1，表示 10%。所有队列的 ApplicationMaster 资源比例上限可通过参数 yarn.scheduler.capacity.maximum-am-resource-percent 设置（可看做默认值），而单个队列可通过参数 yarn.scheduler.capacity.<queue-path>.maximum-am-resource-percent 设置适合自

己的值。

3. 队列访问和权限控制参数

(1) **state** : 队列状态可以为 STOPPED 或者 RUNNING，如果一个队列处于 STOPPED 状态，用户不可以将应用程序提交到该队列或者它的子队列中，类似的，如果 ROOT 队列处于 STOPPED 状态，用户不可以向集群中提交应用程序，但正在运行的应用程序仍可以正常运行结束，以便队列可以优雅地退出。

(2) **acl_submit_applications** : 限定哪些 Linux 用户/用户组可向给定队列中提交应用程序。需要注意的是，该属性具有继承性，即如果一个用户可以向某个队列中提交应用程序，则它可以向它的所有子队列中提交应用程序。配置该属性时，用户之间或用户组之间用“,”分割，用户和用户组之间用空格分割，比如“user1, user2 group1, group2”。

(3) **acl_administer_queue** : 为队列指定一个管理员，该管理员可控制该队列的所有应用程序，比如杀死任意一个应用程序等。同样，该属性具有继承性，如果一个用户可以向某个队列中提交应用程序，则它可以向它的所有子队列中提交应用程序。

一个配置文件实例如下：

```
<configuration>
  <property>
    <name>yarn.scheduler.capacity.maximum-applications</name>
    <value>10000</value>
    <description>最多可同时处于等待和运行状态的应用程序数目</description>
  </property>

  <property>
    <name>yarn.scheduler.capacity.maximum-am-resource-percent</name>
    <value>0.1</value>
    <description>集群中可用于运行 application master 的资源比例上限，这通常用于限制并发运行的应用程序数目。</description>
  </property>

  <property>
    <name>yarn.scheduler.capacity.root.queues</name>
    <value>default</value>
    <description>root 队列的所有子队列，该实例中只有一个</description>
  </property>

  <property>
    <name>yarn.scheduler.capacity.root.default.capacity</name>
    <value>100</value>
    <description>default 队列的资源容量</description>
  </property>

  <property>
    <name>yarn.scheduler.capacity.root.default.user-limit-factor</name>
```

```
<value>1</value>
<description>
  每个用户可使用的资源限制
</description>
</property>
```

```
<property>
  <name>yarn.scheduler.capacity.root.default.maximum-capacity</name>
  <value>100</value>
  <description>
    Default 队列可使用的资源上限.
  </description>
</property>
```

```
<property>
  <name>yarn.scheduler.capacity.root.default.state</name>
  <value>RUNNING</value>
  <description>
    Default 队列的状态，可以是 RUNNING 或者 STOPPED.
  </description>
</property>
```

```
<property>
  <name>yarn.scheduler.capacity.root.default.acl_submit_applications</name>
  <value>*</value>
  <description>
    限制哪些用户可向 default 队列中提交应用程序.
  </description>
</property>
```

```
<property>
  <name>yarn.scheduler.capacity.root.default.acl_administer_queue</name>
  <value>*</value>
  <description>
    限制哪些用户可管理 default 队列中的应用程序，“*”表示任意用户
  </description>
</property>
```

```
<property>
  <name>yarn.scheduler.capacity.node-locality-delay</name>
  <value>-1</value>
  <description>调度器尝试调度一个 rack-local container 之前，最多跳过的调度
    机会，通常而言，该值被设置成集群中机架数目，默认情况下为-1，表示不启
    用该功能。
  </description>
</property>
```

```
</description>
</property>
</configuration>
```

Hadoop 配置文件解析样例

hadoop 配置文件主要有四个，参考默认参数链接：

core-site.xml <http://hadoop.apache.org/docs/r2.2.0/hadoop-project-dist/hadoop-common/core-default.xml>

hdfs-site.xml

<http://hadoop.apache.org/docs/r2.2.0/hadoop-project-dist/hadoop-hdfs/hdfs-default.xml>

mapred-site.xml

<http://hadoop.apache.org/docs/r2.2.0/hadoop-mapreduce-client/hadoop-mapreduce-client-core/mapred-default.xml>

yarn-site.xml

<http://hadoop.apache.org/docs/r2.2.0/hadoop-yarn/hadoop-yarn-common/yarn-default.xml>

下面的配置文件参考 hortonworks.com 网站模版修改，配置文件仅供参考

core-site.xml 配置文件样例

```
[hadoop@rm hadoop]$ cat core-site.xml
```

```
<configuration>
```

```
  <property>
```

```
    <name>hadoop.security.authentication</name>
```

```
    <value>simple</value>
```

```
    <!-- Possible values are simple (no authentication), and kerberos -->
```

```
  </property>
```

```
  <property>
```

```
    <name>ipc.client.connect.max.retries</name>
```

```
    <value>50</value>
```

```
    <!-- Indicates the number of retries a client will make to establish a server connection. -->
```

```
  </property>
```

```
  <property>
```

```
    <name>ipc.client.connection.maxidletime</name>
```

```
    <value>30000</value>
```

```
    <!-- The maximum time in msec after which a client will bring down the connection to the server. -->
```

```
  </property>
```

```
  <property>
```

```

<name>ipc.client.idlethreshold</name>
<value>8000</value>
<!-- Defines the threshold number of connections after which connections will be inspected for idleness. -->
</property>
<property>
<name>io.compression.codecs</name>
<value>org.apache.hadoop.io.compress.GzipCodec,org.apache.hadoop.io.compress.DefaultCodec</value>
<!-- A comma-separated list of the compression codec classes that can be used for compression/decompression. In addition
to any classes specified with this property (which take precedence), codec classes on the classpath are discovered using a Java
ServiceLoader. -->
</property>
<property>
<name>fs.trash.interval</name>
<value>360</value>
<!-- Number of minutes after which the checkpoint gets deleted. If zero, the trash feature is disabled. This option may be
configured both on the server and the client. If trash is disabled server side then the client side configuration is checked. If trash
is enabled on the server side then the value configured on the server is used and the client configuration value is ignored. -->
</property>
<property>
<name>hadoop.security.authorization</name>
<value>>false</value>
<!-- Is service-level authorization enabled? -->
</property>
<property>
<name>fs.defaultFS</name>
<value>hdfs://nn:8020</value>
<!-- The name of the default file system. A URI whose scheme and authority determine the FileSystem implementation. The
uri's scheme determines the config property (fs.SCHEME.impl) naming the FileSystem implementation class. The uri's authority
is used to determine the host, port, etc. for a filesystem. -->
</property>
<property>
<name>hadoop.tmp.dir</name>
<value>/home/dir</value>
<!-- A base for other temporary directories. -->
</property>
<property>
<name>io.file.buffer.size</name>
<value>131072</value>
<!-- The size of buffer for use in sequence files. The size of this buffer should probably be a multiple of hardware page size
(4096 on Intel x86), and it determines how much data is buffered during read and write operations. -->
</property>
</configuration>

```

hdfs-site.xml 配置文件样例

```
[hadoop@rm hadoop]$ cat hdfs-site.xml
<!--Thu Aug 15 20:47:13 2013-->
<configuration>
  <property>
    <name>dfs.block.access.token.enable</name>
    <value>false</value>
    <!-- If "true", access tokens are used as capabilities for accessing datanodes. If "false", no access tokens are checked on
accessing datanodes. -->
  </property>
  <property>
    <name>dfs.datanode.failed.volumes.tolerated</name>
    <value>0</value>
    <!-- The number of volumes that are allowed to fail before a datanode stops offering service. By default any volume failure
will cause a datanode to shutdown. -->
  </property>
  <property>
    <name>dfs.replication.max</name>
    <value>50</value>
    <!-- Maximal block replication.default 512 -->
  </property>
  <property>
    <name>dfs.replication</name>
    <value>3</value>
    <!-- Default block replication. The actual number of replications can be specified when the file is created. The default is used if
replication is not specified in create time. -->
  </property>
  <property>
    <name>dfs.datanode.du.reserved</name>
    <!-- Reserved space in bytes per volume. Always leave this much space free for non dfs use. -->
    <value>1</value>
  </property>
  <property>
    <name>dfs.blockreport.initialDelay</name>
    <value>120</value>
    <!-- Delay for first block report in seconds. -->
  </property>
  <property>
    <name>dfs.datanode.data.dir</name>
    <value>file:///home/dir/dfs/data</value>
    <!-- Determines where on the local filesystem an DFS data node should store its blocks. If this is a comma-delimited list of
```

directories, then data will be stored in all named directories, typically on different devices. Directories that do not exist are ignored. -->

</property>

<property>

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

<value>nn:50070</value>

<!-- The address and the base port where the dfs namenode web ui will listen on. -->

</property>

<property>

<name>dfs.permissions.enabled</name>

<value>true</value>

</property>

<property>

<name>dfs.datanode.ipc.address</name>

<value>0.0.0.0:8010</value>

</property>

<property>

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

<value>file:///home/dir/dfs/name</value>

<!-- Determines where on the local filesystem the DFS name node should store the name table(fsimage). If this is a comma-delimited list of directories then the name table is replicated in all of the directories, for redundancy. -->

</property>

<property>

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

<value>0.0.0.0:8480</value>

</property>

<property>

<name>dfs.heartbeat.interval</name>

<value>3</value>

<!-- Determines datanode heartbeat interval in seconds. -->

</property>

<property>

<name>dfs.namenode.avoid.read.stale.datanode</name>

<value>true</value>

</property>

<property>

<name>dfs.namenode.avoid.write.stale.datanode</name>

<value>true</value>

</property>

<property>

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

<value>750</value>

</property>

<property>

```

<name>fs.permissions.umask-mode</name>
<value>022</value>
</property>
<property>
<name>dfs.datanode.balance.bandwidthPerSec</name>
<value>6250000</value>
</property>
<property>
<name>dfs.namenode.accesstime.precision</name>
<value>0</value>
<!-- The access time for HDFS file is precise upto this value. The default value is 1 hour. Setting a value of 0 disables access
times for HDFS. -->
</property>
<property>
<name>dfs.namenode.stale.datanode.interval</name>
<value>30000</value>
</property>
<property>
<name>dfs.namenode.write.stale.datanode.ratio</name>
<value>1.0f</value>
<!-- When the ratio of number stale datanodes to total datanodes marked is greater than this ratio, stop avoiding writing to
stale nodes so as to prevent causing hotspots. -->
</property>
<property>
<name>dfs.blocksize</name>
<value>134217728</value>
<!-- The default block size for new files, in bytes. You can use the following suffix (case insensitive): k(kilo), m(mega), g(giga),
t(tera), p(peta), e(exa) to specify the size (such as 128k, 512m, 1g, etc.), Or provide complete size in bytes (such as 134217728
for 128 MB). -->
</property>
<property>
<name>dfs.datanode.address</name>
<value>0.0.0.0:50010</value>
</property>
<property>
<name>dfs.datanode.http.address</name>
<value>0.0.0.0:50075</value>
</property>
<property>
<name>dfs.webhdfs.enabled</name>
<value>true</value>
<!-- Enable WebHDFS (REST API) in Namenodes and Datanodes. -->
</property>
<property>

```



```

<name>dfs.namenode.handler.count</name>
<value>100</value>
<!-- The number of server threads for the namenode. -->
</property>
<property>
<name>dfs.permissions.superusergroup</name>
<value>hadoop</value>
</property>
<property>
<name>dfs.namenode.safemode.threshold-pct</name>
<value>1.0f</value>
</property>
</configuration>

```

yarn-site.xml 配置文件样例

yarn-site.xml

```
[hadoop@rm hadoop]$ cat yarn-site.xml
```

```

<configuration>
<property>
<name>yarn.nodemanager.address</name>
<value>0.0.0.0:45454</value>
</property>
<property>
<name>yarn.nodemanager.local-dirs</name>
<value>/home/dir/nm-local-dir</value>
</property>
<property>
<name>yarn.nodemanager.container-monitor.interval-ms</name>
<value>3000</value>
</property>
<property>
<name>yarn.resourcemanager.am.max-attempts</name>
<value>2</value>
</property>
<property>
<name>yarn.nodemanager.health-checker.script.timeout-ms</name>
<value>60000</value>
</property>
<property>
<name>yarn.nodemanager.remote-app-log-dir</name>

```

```
<value>app-logs</value>
</property>
<property>
  <name>yarn.log.server.url</name>
  <value>rm:19888/jobhistory/logs</value>
</property>
<property>
  <name>yarn.resourcemanager.admin.address</name>
  <value>rm:8141</value>
</property>
<!--
<property>
  <name>yarn.scheduler.maximum-allocation-mb</name>
  <value>8192</value>
</property>
<property>
  <name>yarn.scheduler.minimum-allocation-mb</name>
  <value>2048</value>
</property>
<property>
  <name>yarn.nodemanager.resource.memory-mb</name>
  <value>8192</value>
</property>
<property>
  <name>yarn.app.mapreduce.am.resource.mb</name>
  <value>4096</value>
</property>
<property>
  <name>yarn.app.mapreduce.am.command-opts</name>
  <value>3276</value>
</property>-->
<property>
  <name>yarn.nodemanager.vmem-pmem-ratio</name>
  <value>2.1</value>
</property>
<property>
  <name>yarn.nodemanager.delete.debug-delay-sec</name>
  <value>0</value>
</property>
<property>
  <name>yarn.nodemanager.health-checker.interval-ms</name>
  <value>135000</value>
</property>
<property>
```

```
<name>yarn.nodemanager.log.retain-second</name>
<value>604800</value>
</property>
<property>
<name>yarn.resourcemanager.scheduler.class</name>
<value>org.apache.hadoop.yarn.server.resourcemanager.scheduler.capacity.CapacityScheduler</value>
</property>
<property>
<name>yarn.resourcemanager.resource-tracker.address</name>
<value>rm:8025</value>
</property>
<property>
<name>yarn.nodemanager.aux-services</name>
<value>mapreduce_shuffle</value>
</property>
<property>
<name>yarn.nodemanager.log-aggregation.compression-type</name>
<value>gz</value>
</property>
<property>
<name>yarn.nodemanager.health-checker.script.path</name>
<value>${HADOOP_HOME}/etc/hadoop/health_check</value>
</property>
<property>
<name>yarn.nodemanager.container-executor.class</name>
<value>org.apache.hadoop.yarn.server.nodemanager.DefaultContainerExecutor</value>
</property>
<property>
<name>yarn.log-aggregation.retain-seconds</name>
<value>2592000</value>
</property>
<property>
<name>yarn.nodemanager.remote-app-log-dir-suffix</name>
<value>logs</value>
</property>
<property>
<name>yarn.resourcemanager.webapp.address</name>
<value>rm:8088</value>
</property>
<property>
<name>yarn.nodemanager.aux-services.mapreduce.shuffle.class</name>
<value>org.apache.hadoop.mapred.ShuffleHandler</value>
</property>
<property>
```

```

<name>yarn.resourcemanager.address</name>
<value>rm:8050</value>
</property>
<property>
<name>yarn.nodemanager.admin-env</name>
<value>MALLOC_ARENA_MAX=$MALLOC_ARENA_MAX</value>
</property>
<property>
<name>yarn.nodemanager.disk-health-checker.min-healthy-disks</name>
<value>0.25</value>
</property>
<property>
<name>yarn.log-aggregation-enable</name>
<value>true</value>
</property>
<property>
<name>yarn.resourcemanager.scheduler.address</name>
<value>rm:8030</value>
</property>
</configuration>

```

mapred-site.xml 配置文件样例

```

[hadoop@rm hadoop]$ cat mapred-site.xml
<configuration>
<property>
<name>mapreduce.task.timeout</name>
<value>600000</value>
</property>
<property>
<name>mapreduce.cluster.local.dir</name>
<value>/home/dir/mapred/local</value>
<!-- The local directory where MapReduce stores intermediate data files. -->
</property>
<property>
<name>mapreduce.reduce.input.buffer.percent</name>
<value>0.0</value>
</property>
<property>
<name>mapreduce.jobhistory.address</name>
<value>rm:10020</value>

```

```
</property>
<property>
<name>mapreduce.tasktracker.healthchecker.script.path</name>
<value>/home/dir/mapred/jobstatus</value>
<!-- Absolute path to the script which is periodicallyrun by the node health monitoring service to determine if the node is
healthy or not. If the value of this key is empty or the file does not exist in the location configured here, the node health
monitoring service is not started. -->
</property>
<property>
<name>yarn.app.mapreduce.am.staging-dir</name>
<value>/home/dir/tmp/staging</value>
</property>
<!--
<property>
<name>mapreduce.reduce.java.opts</name>
<value>3276</value>
</property>
<property>
<name>mapreduce.map.java.opts</name>
<value>1638</value>
</property>
<property>
<name>mapreduce.map.memory.mb</name>
<value>2048</value>
</property>
<property>
<name>mapreduce.reduce.memory.mb</name>
<value>4096</value>
</property>-->
<property>
<name>mapreduce.jobhistory.intermediate-done-dir</name>
<value>/mr-history/tmp</value>
</property>
<property>
<name>mapreduce.map.sort.spill.percent</name>
<value>0.1</value>
</property>
<property>
<name>mapreduce.jobhistory.done-dir</name>
<value>/mr-history/done</value>
</property>
<property>
<name>mapreduce.jobhistory.webapp.address</name>
<value>rm:19888</value>
```

```
</property>
<property>
  <name>mapreduce.map.speculative</name>
  <value>>false</value>
</property>
<property>
  <name>mapreduce.output.fileoutputformat.compress.type</name>
  <value>BLOCK</value>
</property>
<property>
  <name>mapreduce.reduce.merge.inmem.threshold</name>
  <value>1000</value>
</property>
<property>
  <name>mapreduce.task.io.sort.factor</name>
  <value>100</value>
</property>
<property>
  <name>mapreduce.jobtracker.system.dir</name>
  <value>/home/dir/mapred/system</value>
</property>
<property>
  <name>mapreduce.jobtracker.tasktracker.maxblacklists</name>
  <value>16</value>
</property>
<property>
  <name>mapreduce.reduce.shuffle.input.buffer.percent</name>
  <value>0.7</value>
</property>
<property>
  <name>mapreduce.reduce.shuffle.merge.percent</name>
  <value>0.66</value>
</property>
<property>
  <name>mapred.jobtracker.maxtasks.per.job</name>
  <value>-1</value>
</property>
<property>
  <name>mapreduce.job.reduce.slowstart.completedmaps</name>
  <value>0.05</value>
</property>
<property>
  <name>mapreduce.tasktracker.healthchecker.script.timeout</name>
  <value>60000</value>
```

```
</property>
<property>
  <name>mapreduce.reduce.input.limit</name>
  <value>10737418240</value>
</property>
<property>
  <name>mapreduce.reduce.shuffle.parallelcopies</name>
  <value>30</value>
</property>
<property>
  <name>mapreduce.tasktracker.map.tasks.maximum</name>
  <value>4</value>
</property>
<property>
  <name>mapreduce.reduce.speculative</name>
  <value>false</value>
</property>
<property>
  <name>mapreduce.framework.name</name>
  <value>yarn</value>
</property>
<property>
  <name>mapred.userlog.retain.hours</name>
  <value>24</value>
</property>
</configuration>
```

集群启动及测试

可以批量启动集群中的节点，命令如下：

start-dfs.sh 批量启动 namenode 和 datanode，这类默认还会启动 secondarynamenodes，我们并不需要，建议注释掉如下行
vim sbin/start-dfs.sh

```
#SECONDARY_NAMENODES=$(HADOOP_PREFIX/bin/hdfs getconf -secondarynamenodes 2>/dev/null)
#
# if [ -n "$SECONDARY_NAMENODES" ]; then
#   echo "Starting secondary namenodes [$SECONDARY_NAMENODES]"
#
#   "$HADOOP_PREFIX/sbin/hadoop-daemons.sh" \
#     --config "$HADOOP_CONF_DIR" \
#     --hostnames "$SECONDARY_NAMENODES" \
#     --script "$bin/hdfs" start secondarynamenode
# fi
```

start-yarn.sh 批量启动 resourcemanager 和 nodemanager

启动后可以查看相应的日志，目录在\$HADOOP_HOME/logs 下面，看是否有报错

打开 nn:50070 页面查看

NameNode 'nn:8020' (active)

Started:	Tue Mar 11 20:45:33 CST 2014
Version:	2.2.0, 1529768
Compiled:	2013-10-07T06:28Z by hortonmu from branch-2.2.0
Cluster ID:	CID-5bd0da80-1298-42cb-8237-79aacfc1d21b
Block Pool ID:	BP-1327012523-211.211.211.210-1394460005553

[Browse the filesystem](#)
[NameNode Logs](#)

Cluster Summary

Security is OFF

256 files and directories, 162 blocks = 418 total.

Heap Memory used 27.06 MB is 63% of Committed Heap Memory 42.49 MB. Max Heap Memory is 966.69 MB.

Non Heap Memory used 33.69 MB is 95% of Committed Non Heap Memory 35.13 MB. Max Non Heap Memory is 130 MB.

Configured Capacity	:	506.42 GB			
DFS Used	:	402.37 MB			
Non DFS Used	:	27.09 GB			
DFS Remaining	:	478.94 GB			
DFS Used%	:	0.08%			
DFS Remaining%	:	94.57%			
Block Pool Used	:	402.37 MB			
Block Pool Used%	:	0.08%			
DataNodes usages	:	Min %	Median %	Max %	stdev %
	:	0.06%	0.06%	0.11%	0.03%
Live Nodes	:	3 (Decommissioned: 0)			
Dead Nodes	:	0 (Decommissioned: 0)			

打开 rm:8088

Cluster

About

Nodes

Applications

NEW

NEW SAVING

SUBMITTED

ACCEPTED

RUNNING

REMOVING

FINISHING

FINISHED

FAILED

KILLED

Scheduler

Tools

Cluster Metrics

Apps Submitted	Apps Pending	Apps Running	Apps Completed	Containers Running	Memory Used	Memory Total	Memory Reserved	Active Nodes	Deco
6	0	0	6	0	0 B	24 GB	0 B	3	0

Show 20 entries

Rack	Node State	Node Address	Node HTTP Address	Last health-update	Health-report
/default-rack	RUNNING	nn:45454	nn:8042	11-Mar-2014 23:00:24	
/default-rack	RUNNING	rm:45454	rm:8042	11-Mar-2014 23:00:37	
/default-rack	RUNNING	dn:45454	dn:8042	11-Mar-2014 23:00:36	

Showing 1 to 3 of 3 entries

运行样例测试

teragen 写入数据

hadoop jar .././share/hadoop/mapreduce/hadoop-mapreduce-examples-2.2.0.jar

teragen 1000 /home/input

terasort 然后对数据排序

```
hadoop jar ../../share/hadoop/mapreduce/hadoop-mapreduce-examples-2.2.0.jar  
terasort /home/input /home/output
```

```
14/03/11 23:06:46 INFO Configuration.deprecation: mapred.output.key.class is deprecated. Instead, use  
14/03/11 23:06:46 INFO Configuration.deprecation: mapred.working.dir is deprecated. Instead, use mapr  
14/03/11 23:06:46 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1394539822563_0007  
14/03/11 23:06:47 INFO impl.YarnClientImpl: Submitted application application_1394539822563_0007 to R  
14/03/11 23:06:47 INFO mapreduce.Job: The url to track the job: http://rm:8088/proxy/application_1394  
14/03/11 23:06:47 INFO mapreduce.Job: Running job: job_1394539822563_0007  
14/03/11 23:07:19 INFO mapreduce.Job: Job job_1394539822563_0007 running in uber mode : false  
14/03/11 23:07:19 INFO mapreduce.Job: map 0% reduce 0%  
14/03/11 23:08:14 INFO mapreduce.Job: map 21% reduce 0%  
14/03/11 23:08:17 INFO mapreduce.Job: map 56% reduce 0%  
14/03/11 23:08:20 INFO mapreduce.Job: map 67% reduce 0%  
14/03/11 23:08:29 INFO mapreduce.Job: map 76% reduce 0%  
14/03/11 23:08:32 INFO mapreduce.Job: map 88% reduce 0%  
14/03/11 23:08:34 INFO mapreduce.Job: map 99% reduce 0%
```

```
Total committed heap usage (bytes)=254259200  
Shuffle Errors  
BAD_ID=0  
CONNECTION=0  
IO_ERROR=0  
WRONG_LENGTH=0  
WRONG_MAP=0  
WRONG_REDUCE=0  
File Input Format Counters  
Bytes Read=100000000  
File Output Format Counters  
Bytes Written=100000000  
14/03/11 23:09:19 INFO terasort.TeraSort: done
```

OK！运行没问题。

学习方向

目前的疑惑或问题，及接下来的学习内容计划：

- 1， jobhistory 是干嘛的？在每台机器上启动还是在单台机器上启动好？
- 2， 对于 HDFS 的机制还是不够理解，需要加强原理学习，配置文件也需要进一步改善，对于 HDFS 这一块主要存在一下一些问题：
 - web 界面无法浏览文件 dfs.webhdfs.enabled 选项已开启
 - heap memory 含义及在哪里自定义配置？
 - HDFS NFS Gateway 配置
 - HDFS 高可用的配置 High Availability With QJM 及 High Availability With NFS
 - HDFS Federation 的配置
 - HDFS HDFS Snapshots 了解
 - HDFS 的架构深入学习
 - 学习网站：<http://hadoop.apache.org> 官网中 HDFS 模块
- 3， 对 YARN 的资源管理理解不够，需要加强学习
- 4， 对于内存分配理解不够，其实这也是对 YARN 运行过程不理解导致的
以上内容可以通过一下几个网站去学习：

<http://hadoop.apache.org>

<http://dongxicheng.org/category/mapreduce-nextgen/page/2/>

<http://hortonworks.com/products/hdp-2/?b=2#documentation>

5, 上面的学习仅仅局限与 hadoop 本身, 还有 hadoop 生态系统中的其他内容需要学习, 主要有官网项目中的如下这些:

- [Ambari™](#): A web-based tool for provisioning, managing, and monitoring Apache Hadoop clusters which includes support for Hadoop HDFS, Hadoop MapReduce, Hive, HCatalog, HBase, ZooKeeper, Oozie, Pig and Sqoop. Ambari also provides a dashboard for viewing cluster health such as heatmaps and ability to view MapReduce, Pig and Hive applications visually alongwith features to diagnose their performance characteristics in a user-friendly manner.
- [Avro™](#): A data serialization system.
- [Cassandra™](#): A scalable multi-master database with no single points of failure.
- [Chukwa™](#): A data collection system for managing large distributed systems.
- [HBase™](#): A scalable, distributed database that supports structured data storage for large tables.
- [Hive™](#): A data warehouse infrastructure that provides data summarization and ad hoc querying.
- [Mahout™](#): A Scalable machine learning and data mining library.
- [Pig™](#): A high-level data-flow language and execution framework for parallel computation.
- [Spark™](#): A fast and general compute engine for Hadoop data. Spark provides a simple and expressive programming model that supports a wide range of applications, including ETL, machine learning, stream processing, and graph computation.
- [ZooKeeper™](#): A high-performance coordination service for distributed applications.

nodemanager 上报自身资源

```
INFO org.apache.hadoop.yarn.server.resourcemanager.rmnode.RMNodeImpl: dn:45454 Node Transitioned from NEW to RUNNING
INFO org.apache.hadoop.yarn.server.resourcemanager.scheduler.capacity.CapacityScheduler: Added node dn:45454 clusterResource: <memory:8192, vCores:8>
INFO org.apache.hadoop.yarn.util.RackResolver: Resolved rm to /default-rack
INFO org.apache.hadoop.yarn.server.resourcemanager.ResourceTrackerService: NodeManager from node rm(cmPort: 45454 httpPort: 8042) registered with capability: <memory:8192, vCores:8>
nodeId rm:45454
INFO org.apache.hadoop.yarn.server.resourcemanager.rmnode.RMNodeImpl: rm:45454 Node Transitioned from NEW to RUNNING
INFO org.apache.hadoop.yarn.server.resourcemanager.scheduler.capacity.CapacityScheduler: Added node rm:45454 clusterResource: <memory:16384, vCores:16>
INFO org.apache.hadoop.yarn.util.RackResolver: Resolved nn to /default-rack
INFO org.apache.hadoop.yarn.server.resourcemanager.rmnode.RMNodeImpl: nn:45454 Node Transitioned from NEW to RUNNING
INFO org.apache.hadoop.yarn.server.resourcemanager.scheduler.capacity.CapacityScheduler: Added node nn:45454 clusterResource: <memory:24576, vCores:24>
INFO org.apache.hadoop.yarn.server.resourcemanager.ResourceTrackerService: NodeManager from node nn(cmPort: 45454 httpPort: 8042) registered with capability: <memory:24576, vCores:24>
nodeId nn:45454
```

源码编译环境配置

需要的软件

- Maven 3.0.4
- Java 1.6.0_65
- Hadoop 2.2.0
- gcc++
- cmake
- protoc
- **zlib-devel**
- **openssl-devel** 这两个必须安装，否则会导致报错
- **yum -y install lzo-devel zlib-devel gcc autoconf automake libtool**

安装所需软件

1、checkout 源码

svn co <https://svn.apache.org/repos/asf/hadoop/common/tags/release-2.2.0/>

2、安装 protoc

到 Protocol Buffers 的官网 <https://code.google.com/p/protobuf/> 下载 2.5.0 的安装源文件进行安装：

```
tar -zxf protobuf-2.5.0.tar.gz
cd protobuf-2.5.0
./configure --prefix=/usr/local/protobuf
make check
```

make install

安装完成后，执行 `protoc -v` 验证是否安装成功。

3. 安装 cmake gcc++

官方（<http://www.cmake.org/cmake/resources/software.html>）下载对应系统的版本进行安装

centos 直接执行 `yum install gcc++ cmake`

4. 安装 maven

下载 <http://mirrors.cnnic.cn/apache/maven/maven-3/3.2.1/binaries/apache-maven-3.2.1-bin.tar.gz>

解压到相应目录，添加环境变量

```
export MAVEN_HOME=/usr/lib/apache-maven-3.2.1/
```

```
export PATH=$MAVEN_HOME/bin:$PATH
```

5. 执行 mvn 命令进行编译了

```
mvn clean package -Pdist,native -DskipTests -Dtar
```

常见错误

```
ERROR] Failed to execute goal org.apache.hadoop:hadoop-maven-plugins:3.0.0-
SNAPSHOT:protoc (compile-protoc) on project hadoop-common:
org.apache.maven.plugin.MojoExecutionException: 'protoc -version' did not return
a version -> [Help 1]
```

- 上面的错误原因是：没有安装 `protoc` 或者没有安装 `zlib-devel`、`openssl-devel`

```
[ERROR] Failed to execute goal org.apache.maven.plugins:maven-antrun-
plugin:1.7:run (make) on project hadoop-common: An Ant BuildException has
occured: Execute failed: java.io.IOException: Cannot run program "cmake" (in
directory "/Users/micmiu/no_sync/opensource_code/hadoop/trunk/hadoop-
common-project/hadoop-common/target/native"): error=2, No such file or
directory
[ERROR] around Ant part ...<exec
dir="/Users/micmiu/no_sync/opensource_code/hadoop/trunk/hadoop-common-
project/hadoop-common/target/native" executable="cmake"
failonerror="true">... @ 4:154 in
/Users/micmiu/no_sync/opensource_code/hadoop/trunk/hadoop-common-
project/hadoop-common/target/antrun/build-main.xml
```

上面的错误原因是：没有安装 `cmake`

```
ERROR] Failed to execute goal org.apache.maven.plugins:maven-antrun-
plugin:1.7:run (make) on project hadoop-common: An Ant BuildException has
occured: exec returned: 2
[ERROR] around Ant part ...<exec
dir="/Users/micmiu/no_sync/opensource_code/hadoop/trunk/hadoop-common-
project/hadoop-common/target/native" executable="make" failonerror="true">...
@ 7:153 in /Users/micmiu/no_sync/opensource_code/hadoop/trunk/hadoop-
common-project/hadoop-common/target/antrun/build-main.xml
[ERROR] -> [Help 1]
org.apache.maven.lifecycle.LifecycleExecutionException: Failed to execute goal
org.apache.maven.plugins:maven-antrun-plugin:1.7:run (make) on project
hadoop-common: An Ant BuildException has occured: exec returned: 2
around Ant part ...<exec
dir="/Users/micmiu/no_sync/opensource_code/hadoop/trunk/hadoop-common-
project/hadoop-common/target/native" executable="make" failonerror="true">...
@ 7:153 in /Users/micmiu/no_sync/opensource_code/hadoop/trunk/hadoop-
common-project/hadoop-common/target/antrun/build-main.xml
at
org.apache.maven.lifecycle.internal.MojoExecutor.execute(MojoExecutor.java:216)
at
org.apache.maven.lifecycle.internal.MojoExecutor.execute(MojoExecutor.java:153)
at
org.apache.maven.lifecycle.internal.MojoExecutor.execute(MojoExecutor.java:145)
at
org.apache.maven.lifecycle.internal.LifecycleModuleBuilder.buildProject(Lifecycle
ModuleBuilder.java:84)
at
org.apache.maven.lifecycle.internal.LifecycleModuleBuilder.buildProject(Lifecycle
ModuleBuilder.java:59)
at
org.apache.maven.lifecycle.internal.LifecycleStarter.singleThreadedBuild(Lifecycle
Starter.java:183)
at
org.apache.maven.lifecycle.internal.LifecycleStarter.execute(LifecycleStarter.java:
161)
at org.apache.maven.DefaultMaven.doExecute(DefaultMaven.java:317)
at org.apache.maven.DefaultMaven.execute(DefaultMaven.java:152)
at org.apache.maven.cli.MavenCli.execute(MavenCli.java:555)
at org.apache.maven.cli.MavenCli.doMain(MavenCli.java:214)
dir="/Users/micmiu/no_sync/opensource_code/hadoop/trunk/hadoop-common-
project/hadoop-common/target/native" executable="make" failonerror="true">...
@ 7:153 in /Users/micmiu/no_sync/opensource_code/hadoop/trunk/hadoop-
common-project/hadoop-common/target/antrun/build-main.xml
at org.apache.maven.plugin.antrun.AntRunMojo.execute(AntRunMojo.java:355)
```

```
at
org.apache.maven.plugin.DefaultBuildPluginManager.executeMojo(DefaultBuildPlu
ginManager.java:106)
at
org.apache.maven.lifecycle.internal.MojoExecutor.execute(MojoExecutor.java:208)
... 19 more
Caused by: /Users/micmiu/no_sync/opensource_code/hadoop/trunk/hadoop-
common-project/hadoop-common/target/antrun/build-main.xml:7: exec returned:
2
```

参考：<http://www.micmiu.com/bigdata/hadoop/hadoop-build-source-2-2-0/>

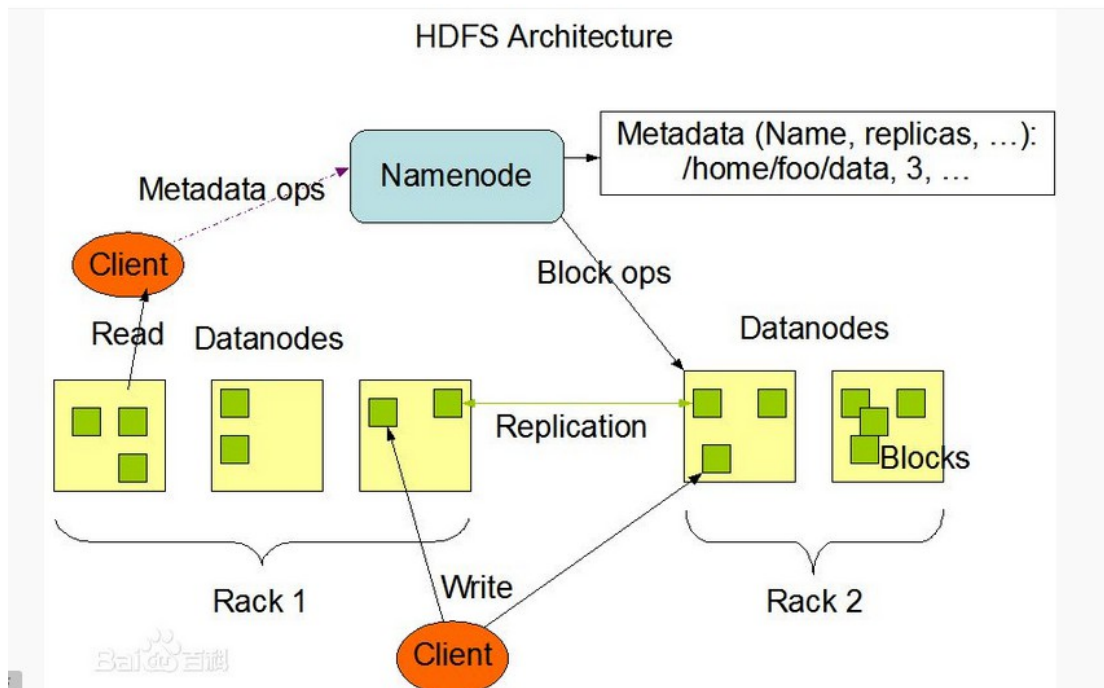
HDFS HA and Federation

HA

目前 HA 有两种方案：

- High Availability With QJM
- High Availability With NFS

hdfs 原理图：



High Availability With QJM

实验环境：

```
nn namenode 211.211.211.210
nn1 namenode 211.211.211.209
rm resourcemanager 211.211.211.211
dn datanode 211.211.211.212
```

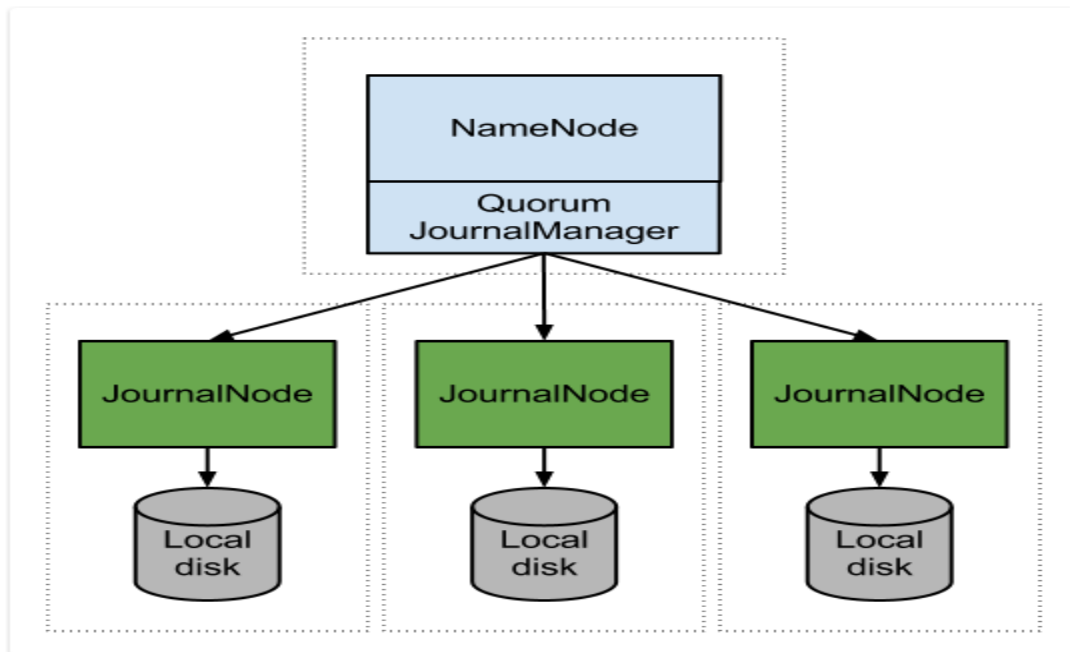
参考官方文档：

<http://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/HDFSHighAvailabilityWithQJM.html>

<http://yanbohappy.sinaapp.com/?p=205>

QJM 的基本原理就是用 $2N+1$ 台 JournalNode 存储 EditLog，每次写数据操作有大多数 ($\geq N+1$) 返回成功时即认为该次写成功，数据不会丢失了。当然这个算法所能容忍的是最多有 N 台机器挂掉，如果多于 N 台挂掉，这个算法就失效了。这个原理是基于 Paxos 算法的，可以参考 [http://en.wikipedia.org/wiki/Paxos_\(computer_science\)](http://en.wikipedia.org/wiki/Paxos_(computer_science))。

基本原理图：



第一步：zookeeper 集群配置

官方参考：<http://zookeeper.apache.org/doc/r3.4.5/index.html>

wget <http://mirrors.cnnic.cn/apache/zookeeper/stable/zookeeper-3.4.5.tar.gz>

解压到相关目录，这里只配置 2 台机器，就配置在两台 namenode 上

解压后修改 conf/ zoo.cfg 文件，内容如下：

修改配置文件：

tickTime=2000

initLimit=10

syncLimit=5

dataDir=/home/dir/zookeeper

clientPort=2181

server.1=nn:2888:3888

server.2=nn1:2888:3888

server.3=rm:2888:3888

当然也可以多加几台，最好是奇数台，可以参考官方

在我们配置的 dataDir 指定的目录下面，创建一个 myid 文件，里面内容为一个数字，用来标识当前主机，conf/zoo.cfg 文件中配置的 server.X 中 X 为什么数字，则 myid 文件中就输入这个数字，例如：

nn:echo "1" > /home/dir/zookeeper/myid

nn1:echo "2" > /home/dir/zookeeper/myid


```
rm:echo "3" > /home/dir/zookeeper/myid
```

在三台机器上分别启动 zookeeper：

```
zkServer.sh start
```

第二步：hdfs-core.xml 参数配置（支持 QJM）

主要涉及的配置文件有 core-site.xml 和 hdfs-core.xml，可以参考官方的说明中的 Deployment 部分

<http://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/HDFSHighAvailabilityWithQJM.html#Deployment>

我这里给出配置文件（都是在原来配置文件基础上增加）：

```
<!-- HDFS HA use QJM config -->
<property>
  <name>dfs.nameservices</name>
  <value>jimmycluster</value>
</property>
<property>
  <name>dfs.ha.namenodes.jimmycluster</name>
  <value>nn,nn1</value>
</property>
<property>
  <name>dfs.namenode.rpc-address.jimmycluster.nn</name>
  <value>nn:9000</value>
</property>
<property>
  <name>dfs.namenode.rpc-address.jimmycluster.nn1</name>
  <value>nn1:9000</value>
</property>
<property>
  <name>dfs.namenode.http-address.jimmycluster.nn</name>
  <value>nn:9001</value>
</property>
<property>
  <name>dfs.namenode.http-address.jimmycluster.nn1</name>
  <value>nn1:9001</value>
</property>
<property>
  <name>dfs.namenode.shared.edits.dir</name>
  <value>qjournal://nn:8485;dn:8485;nn1:8485/jimmycluster</value>
```

```

</property>
<property>
  <name>dfs.client.failover.proxy.provider.jimmycluster</name>
<value>org.apache.hadoop.hdfs.server.namenode.ha.ConfiguredFailoverProxyProvid
er</value>
</property>
<property>
  <name>dfs.ha.fencing.methods</name>
  <value>sshfence</value>
</property>
<property>
  <name>dfs.ha.fencing.ssh.private-key-files</name>
  <value>/home/hadoop/.ssh/id_rsa</value>
</property>
<property>
  <name>dfs.journalnode.edits.dir</name>
  <value>/home/dir/journal/node/local/data</value>
</property>
<property>
  <name>dfs.ha.automatic-failover.enabled</name>
  <value>true</value>
</property>

```

同时注释掉如下行：

```

<!--
<property>
  <name>dfs.namenode.http-address</name>
  <value>nn:50070</value>
  <!-- The address and the base port where the dfs namenode web ui will listen on.
-->
</property>
-->

```

第三步：core-site.xml 配置

```

</property>
<property>
  <name>fs.defaultFS</name>
  <value>hdfs://jimmycluster</value>
</property>

```

```
<property>
  <name>ha.zookeeper.quorum</name>
  <value>nn:2181,nn1:2181</value>
</property>
```

第四步：HA 启动过程

我们这里搭建的是一个全新的集群，清楚以前相关的数据

```
rm -rf /home/app/hadoop-2.2.0/logs/*
rm -rf /home/dir/dfs/name/*
rm -rf /home/dir/dfs/data/*
rm -rf /home/dir/journal/node/local/data/*
初始化 zkfc
```

```
hdfs zkfc -formatZK
```

```
y
14/03/14 20:44:10 INFO ha.ActiveStandbyElector: Recursively deleting /hadoop-ha/jimmycluster from ZK...
14/03/14 20:44:10 INFO ha.ActiveStandbyElector: Successfully deleted /hadoop-ha/jimmycluster from ZK.
14/03/14 20:44:11 INFO ha.ActiveStandbyElector: Successfully created /hadoop-ha/jimmycluster in ZK.
14/03/14 20:44:11 INFO zookeeper.ZooKeeper: Session: 0x144c091cbfc0000 closed
14/03/14 20:44:11 INFO zookeeper.ClientCnxn: EventThread shut down
```

启动集群中所有的 journalnode

我这里配置的是 nn,nn1,dn 三台，分别执行

```
hadoop-daemon.sh start journalnode
```

格式化 HDFS 文件系统

在其中一台 namenode 上执行：

```
hdfs namenode -format
```

将格式化后端相关数据传输到另一台上：

```
scp -r /home/dir/dfs/name/* nn1:/home/dir/dfs/name/
```

```
[hadoop@nn hadoop]$ scp -r /home/dir/dfs/name/* nn1:/home/dir/dfs/name/
fsimage_00000000000000000000
fsimage_00000000000000000000.md5
seen_txid
VERSION
```

如果没有 editlog 就不需要传这些，直接执行 `hdfs namenode -bootstrapStandby` 同步元数据

启动 namenode：hadoop-daemon.sh start namenode

```
14/03/29 12:35:25 INFO namenode.NameNode: registered UNIX signal handlers for [TERM, HUP, INT]
14/03/29 12:35:27 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable

About to bootstrap Standby ID nn2 from:
  Nameservice ID: jimmycluster
  Other Namenode ID: nn1
  Other NN's HTTP address: nn1:9001
  Other NN's IPC address: nn1/211.211.211.100:9000
  Namespace ID: 1437884384
  Block pool ID: BP-2028081358-211.211.211.100-1396067566229
  Cluster ID: CID-ef4b940f-ba36-4ff6-a5ae-c0be4fa833ce
  Layout version: -47

14/03/29 12:35:30 INFO common.Storage: Storage directory /home/dir/dfs/name has been successfully formatted.
14/03/29 12:35:30 INFO namenode.TransferFsImage: Opening connection to http://nn1:9001/getimage?getimage=1&txid=0&storageInfo=-47:1437884384:0:CID-ef4b940f-ba36-4ff6-a5ae-c0be4fa833ce
14/03/29 12:35:31 INFO namenode.TransferFsImage: Transfer took 0.63s at 0.00 KB/s
14/03/29 12:35:31 INFO namenode.TransferFsImage: Downloaded file fsimage.ckpt_00000000000000000000 size 194 bytes.
14/03/29 12:35:31 INFO util.ExitUtil: Exiting with status 0
```

启动 zkfc

如果说到此步直接执行 start-dfs.sh 启动其他所以节点，有可能会报导致 namenode 报错，所以建议还是先启动 zkfc，两台 namenode 上执行：

`hadoop-daemon.sh start zkfc`

启动集群的其他节点

start-dfs.sh
start-yarn.sh

查看日志看是否有报错

tail -100f /home/app/hadoop-2.2.0/logs/hadoop-hadoop-namenode-nn1.log

....

我们目前启动是 namenode 和 datanode 及 journalnode，执行 jps

```
[hadoop@nn hadoop]$ jps
16234 JournalNode
16662 DataNode
16573 NameNode
16992 DFSZKFailoverController
17145 Jps
15664 QuorumPeerMain
```

第五步：查看机器 HA 状态

命令行查看：`hdfs haadmin -getServiceState { nn or nn1 }`

```
[hadoop@nn hadoop]$ hdfs haadmin -getServiceState nn
14/03/14 20:59:45 WARN util.NativeCodeLoader: Unable to load native-hadoop library from classpath; using native implementation
active
[hadoop@nn hadoop]$ hdfs haadmin -getServiceState nn1
14/03/14 20:59:52 WARN util.NativeCodeLoader: Unable to load native-hadoop library from classpath; using native implementation
standby
```

网页方式查看，访问：<http://nn:9001> and <http://nn1:9001>



NameNode 'nn:9000' (active)

Started:	Fri Mar 14 20:54:59 CST 2014
Version:	2.2.0, 1529768
Compiled:	2013-10-07T06:28Z by hortonmu from branch-2.2.0
Cluster ID:	CID-3e35b0fc-6e8a-4d6a-97ce-edc58ab282dc
Block Pool ID:	BP-448603857-211.211.211.210-1394801413058

[Browse the filesystem](#)
[NameNode Logs](#)

Cluster Summary

Security is *OFF*



NameNode 'nn1:9000' (standby)

Started:	Fri Mar 14 20:55:05 CST 2014
Version:	2.2.0, 1529768
Compiled:	2013-10-07T06:28Z by hortonmu from branch-2.2.0
Cluster ID:	CID-3e35b0fc-6e8a-4d6a-97ce-edc58ab282dc
Block Pool ID:	BP-448603857-211.211.211.210-1394801413058

[NameNode Logs](#)

Cluster Summary

Security is *OFF*

可以新建目录，上传数据测试，看 hdfs 机器是否运行正常

```
hdfs dfs -mkdir /app
```

```
hdfs dfs -put /home/* /app
```

第六步：failover 测试

我们这里直接 kill -9 namenodepid 来测试

```
[hadoop@nn hadoop]$ kill -9 `cat /tmp/hadoop-hadoop-namenode.pid`
```

```
[hadoop@nn hadoop]$ hdfs haadmin -getServiceState nn
```

failed

```
[hadoop@nn hadoop]$ hdfs haadmin -getServiceState nn1
```

active

```
[hadoop@nn hadoop]$ jps
```

16234 JournalNode

16662 DataNode

17977 Jps

16992 DFSZKFailoverController

15664 QuorumPeerMain

```
15664 QuorumPeerMain
[hadoop@nn hadoop]$ kill -9 `cat /tmp/hadoop-hadoop-namenode.pid`
[hadoop@nn hadoop]$ hdfs haadmin -getServiceState nn
14/03/14 21:09:19 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using
14/03/14 21:09:20 INFO ipc.Client: Retrying connect to server: nn/211.211.211.210:9000. Already tried 0 time(s)
SECONDS)
Operation failed: Call From nn/211.211.211.210 to nn:9000 failed on connection exception: java.net.ConnectExcep
onnectionRefused
[hadoop@nn hadoop]$ hdfs haadmin -getServiceState nn1
14/03/14 21:09:28 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using
active
[hadoop@nn hadoop]$ jps
16234 JournalNode
16662 DataNode
17977 Jps
16992 DFSZKFailoverController
15664 QuorumPeerMain
```



NameNode 'nn1:9000' (active)

Started:	Fri Mar 14 20:55:05 CST 2014
Version:	2.2.0, 1529768
Compiled:	2013-10-07T06:28Z by hortonmu from branch-2.2.0
Cluster ID:	CID-3e35b0fc-6e8a-4d6a-97ce-edc58ab282dc
Block Pool ID:	BP-448603857-211.211.211.210-1394801413058

[Browse the filesystem](#)

[NameNode Logs](#)

Cluster Summary

Security is *OFF*

nn1 已经自动切换到 active 了，可以写入数据试试看，测试完成启动 nn 上的 namenode

zkfc 切换日志

2014-03-31 06:10:22,911 WARN org.apache.hadoop.ha.FailoverController: Unable to gracefully make NameNode at nn2/211.211.211.99:9000 standby (unable to connect)

java.net.ConnectException: Call From nn1.cc/211.211.211.100 to nn2:9000 failed on connection exception: java.net.ConnectException: Connection refused; For more details see: <http://wiki.apache.org/hadoop/ConnectionRefused>

at sun.reflect.NativeConstructorAccessorImpl.newInstance0(Native Method)

.....

2014-03-31 06:10:28,885 INFO org.apache.hadoop.ha.SshFenceByTcpPort.jsch: Next authentication method: publickey

2014-03-31 06:10:29,229 INFO org.apache.hadoop.ha.SshFenceByTcpPort.jsch: Authentication succeeded (publickey).

2014-03-31 06:10:29,232 INFO org.apache.hadoop.ha.SshFenceByTcpPort: Connected to nn2

2014-03-31 06:10:29,372 INFO org.apache.hadoop.ha.SshFenceByTcpPort: Looking for process running on port 9000

2014-03-31 06:10:31,038 INFO org.apache.hadoop.ha.SshFenceByTcpPort: Indeterminate response from trying to kill service. Verifying whether it is running using nc...

2014-03-31 06:10:31,111 WARN org.apache.hadoop.ha.SshFenceByTcpPort: nc -z nn2 9000 via ssh: bash: nc: command not found

2014-03-31 06:10:31,111 INFO org.apache.hadoop.ha.SshFenceByTcpPort: Verified that the service is down.

2014-03-31 06:10:31,111 INFO org.apache.hadoop.ha.SshFenceByTcpPort.jsch: Disconnecting from nn2 port 22

2014-03-31 06:10:31,120 INFO org.apache.hadoop.ha.NodeFencer: ===== Fencing successful by method org.apache.hadoop.ha.SshFenceByTcpPort(null) =====

2014-03-31 06:10:31,120 INFO org.apache.hadoop.ha.ActiveStandbyElector: Writing znode /hadoop-ha/jimmycluster/ActiveBreadCrumb to indicate that the local node is the most recent active...

2014-03-31 06:10:31,128 INFO org.apache.hadoop.ha.SshFenceByTcpPort.jsch: Caught an exception, leaving main loop due to Socket closed

2014-03-31 06:10:31,366 INFO org.apache.hadoop.ha.ZKFailoverController: Trying to make NameNode at nn1/211.211.211.100:9000 active...

2014-03-31 06:10:53,295 INFO org.apache.hadoop.ha.ZKFailoverController: Successfully transitioned NameNode at nn1/211.211.211.100:9000 to active state

似乎需要 nc 命令的支持

后续出现的问题

1, 当 HA 的 dfs.ha.fencing.methods 做如下配置时

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

如果 active 直接断网了, 就会导致无法顺利的切换, 报错如下:

```
2014-03-31 06:57:24,276 INFO org.apache.hadoop.ha.ActiveStandbyElector: Trying
to re-establish ZK session
2014-03-31 06:57:24,286 INFO org.apache.zookeeper.ZooKeeper: Session:
0x24514de1d140038 closed
2014-03-31 06:57:25,287 INFO org.apache.zookeeper.ZooKeeper: Initiating client
connection, connectString=nn1:2181,nn2:2181,rm:2181 sessionTimeout=5000
watcher=org.apache.hadoop.ha.ActiveStandbyElector$WatcherWithClientRef@2d63
62fc
2014-03-31 06:57:25,290 INFO org.apache.zookeeper.ClientCnxn: Opening socket
connection to server rm.cc/211.211.211.130:2181. Will not attempt to authenticate
using SASL (Unable to locate a login configuration)
2014-03-31 06:57:25,291 INFO org.apache.zookeeper.ClientCnxn: Socket connection
established to rm.cc/211.211.211.130:2181, initiating session
2014-03-31 06:57:25,298 INFO org.apache.zookeeper.ClientCnxn: Session
establishment complete on server rm.cc/211.211.211.130:2181, sessionId =
0x34514de1d850040, negotiated timeout = 5000
2014-03-31 06:57:25,300 INFO org.apache.zookeeper.ClientCnxn: EventThread shut
down
2014-03-31 06:57:25,300 INFO org.apache.hadoop.ha.ActiveStandbyElector: Session
connected.
2014-03-31 06:57:25,310 INFO org.apache.hadoop.ha.ActiveStandbyElector:
Checking for any old active which needs to be fenced...
2014-03-31 06:57:25,313 INFO org.apache.hadoop.ha.ActiveStandbyElector: Old
node exists:
0a0c6a696d6d79636c757374657212036e6e311a036e6e3120a84628d33e
2014-03-31 06:57:25,318 INFO org.apache.hadoop.ha.ZKFailoverController: Should
fence: NameNode at nn1/211.211.211.100:9000
2014-03-31 06:57:28,261 INFO org.apache.hadoop.ipc.Client: Retrying connect to
server: nn1/211.211.211.100:9000. Already tried 0 time(s); retry policy is
```


RetryUpToMaximumCountWithFixedSleep(maxRetries=1, sleepTime=1 SECONDS)
2014-03-31 06:57:31,251 WARN org.apache.hadoop.ha.FailoverController: Unable to gracefully make NameNode at nn1/211.211.211.100:9000 standby (unable to connect)
java.net.NoRouteToHostException: No Route to Host from nn2.cc/211.211.211.99 to nn1:9000 failed on socket timeout exception: java.net.NoRouteToHostException: No route to host; For more details see: <http://wiki.apache.org/hadoop/NoRouteToHost>

持续报这个错，一直在确认 SSH，但为什么没有 timeout 配置呢？不知道为什么不切换？？

High Availability With NFS

此方案和上面的基于 QJM 方案差不多，只是方法改为了共享存储，去掉 journalnode 节点，配置更改也比较小。

参考官方文档：

<http://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/HDFSHighAvailabilityWithNFS.html>

环境和上面一样，停掉集群，NFS 方案也需要依赖 zookeeper 协调系统，所以这个保持运行。

保持 **zookeeper** 集群的运行

更改 **hdfs-core.xml** 配置文件

```
<property>
  <name>dfs.nameservices</name>
  <value>jimmycluster</value>
</property>
<property>
  <name>dfs.ha.namenodes.jimmycluster</name>
  <value>nn,nn1</value>
</property>
<property>
  <name>dfs.namenode.rpc-address.jimmycluster.nn</name>
  <value>nn:9000</value>
</property>
</property>
```

```

    <name>dfs.namenode.rpc-address.jimmycluster.nn1</name>
    <value>nn1:9000</value>
  </property>
</property>
    <name>dfs.namenode.http-address.jimmycluster.nn</name>
    <value>nn:9001</value>
  </property>
</property>
    <name>dfs.namenode.http-address.jimmycluster.nn1</name>
    <value>nn1:9001</value>
  </property>
</property>
    <name>dfs.namenode.shared.edits.dir</name>
    <value>file:///home/dir/nfs/name</value>
  </property>
</property>
    <name>dfs.client.failover.proxy.provider.jimmycluster</name>
<value>org.apache.hadoop.hdfs.server.namenode.ha.ConfiguredFailoverProxyProvid
er</value>
  </property>
</property>
    <name>dfs.ha.fencing.methods</name>
    <value>sshfence</value>
  </property>
</property>
    <name>dfs.ha.fencing.ssh.private-key-files</name>
    <value>/home/hadoop/.ssh/id_rsa</value>
  </property>
</property>
    <name>dfs.journalnode.edits.dir</name>
    <value>/home/dir/journal/node/local/data</value>
  </property>
</property>
    <name>dfs.ha.automatic-failover.enabled</name>
    <value>true</value>
  </property>
</property>

```

同时注释掉 namenode 的元数据存储位置：

```

<!--
<property>
  <name>dfs.namenode.name.dir</name>
  <value>file:///home/dir/nfs/name</value>
</property>
-->

```

OK , 初始化 zkfc

```
hdfs zkfc -formatZK
```

格式化 HDFS 文件系统

在其中一台 namenode 上执行：

```
hdfs namenode -format
```

启动集群：

```
start-dfs.sh
```

检查日志看是否有报错，没有报错，但启动完后两个都是 active 奇怪，

测试参照前面一节。

单台 namenode 扩展 standbay

在现有集群上扩展一台 standbay 实验不成功，后来检查主要出错在两个地方，一个是 datanode 存储位置改了，另一个是有些命令执行理解错误

我们这里基于原来的架构增加一台 nn1 作为 namenode 的热备份

```
nn namenode 211.211.211.210
nn1 namenode 211.211.211.209 增加
rm resourcemanager 211.211.211.211
dn datanode 211.211.211.212
```

第一步：停止集群

```
stop-dfs.sh
```

第二步：修改配置文件，主要是 hdfs-core.xml core-site.xml

hdfs-core.xml：

参照上面的基于 QJM 方案的配置文件，但需要注意的是 **datanode 的存储位置**和**先前集群需要一致**，否则 namenode 会无法找到 block，排查这个错误纠结了我好久。

core-site.xml：参考前面

一样参考前面

将两个配置文件更新到所有的 **datanode** 上

将 namenode 的安装文件拷贝到另一台 standby 上

第三步：搭建 zookeeper 集群

我们这里试着搭建 3 台测试：

修改配置文件：

tickTime=2000

initLimit=10

syncLimit=5

dataDir=/home/dir/zookeeper

clientPort=2181

server.1=nn:2888:3888

server.2=nn1:2888:3888

server.3=rm:2888:3888

当然也可以多加几台，最好是奇数台，可以参考官方

在我们配置的 dataDir 指定的目录下面，创建一个 myid 文件，里面内容为一个数字，用来标识当前主机，conf/zoo.cfg 文件中配置的 server.X 中 X 为什么数字，则 myid 文件中就输入这个数字，例如：

nn:echo "1" > /home/dir/zookeeper/myid

nn1:echo "2" > /home/dir/zookeeper/myid

rm:echo "3" > /home/dir/zookeeper/myid

在三台机器上分别启动 zookeeper：

zkServer.sh start

查看 zookeeper 的状态，其中有一台会是 leader

```
[hadoop@nn conf]$ zkServer.sh status
JMX enabled by default
Using config: /home/app/zookeeper-3.4.5/bin/../conf/zoo.cfg
Mode: follower
```

```
[hadoop@nn1 app]$ zkServer.sh status
JMX enabled by default
Using config: /home/app/zookeeper-3.4.5/bin/../conf/zoo.cfg
Mode: leader
```

第四步：启动 journalnode

所有 journalnode 节点执行：

Hadoop-damon.sh start journalnode

第五步：启动 zkfc

在两台 namenode 上启动：

```
hdfs zkfc -formatZK
Hadoop-damon.sh start zkfc
```

第六步：启动原 namenode

nn 上执行：

```
Hadoop-damon.sh start namenode
```

第六步：在 nn1 上执行如下命令

格式化 namenode：

```
hdfs namenode -bootstrapStandby
```

将 name 目录下的元数据和 edit_log 传输到 nn1 上：

```
scp -r /home/dir/dfs/name/* nn1:/home/dir/dfs/name/
```

初始化 JournalNodes：

```
hdfs namenode -initializeSharedEdits
```

上面三条命令的执行顺序不能反了

第八步：启动 nn1 上的 namenode

```
Hadoop-damon.sh start namenode
```

```
2014-03-22 20:59:59,716 INFO org.apache.zookeeper.ClientCnxn: Socket connection established to nn1/211.211.211.209:2181, initiating session
2014-03-22 20:59:59,729 INFO org.apache.zookeeper.ClientCnxn: Session establishment complete on server nn1/211.211.211.209:2181, sessionId = 0x244e9d0d0510002, negotiated timeout = 5000
2014-03-22 20:59:59,816 INFO org.apache.hadoop.ha.ActiveStandbyElector: Session connected.
2014-03-22 20:59:59,831 INFO org.apache.hadoop.ha.ZKFailoverController: ZK Election indicated that NameNode at nn1/211.211.211.209:9000 should become standby
2014-03-22 20:59:59,872 INFO org.apache.hadoop.ha.ZKFailoverController: Successfully transitioned NameNode at nn1/211.211.211.209:9000 to standby state
```

```
2014-03-22 20:58:43,780 INFO org.apache.hadoop.ha.ActiveStandbyElector: Checking for any old active which needs to be fenced...
2014-03-22 20:58:43,805 INFO org.apache.hadoop.ha.ActiveStandbyElector: No old node to fence
2014-03-22 20:58:43,805 INFO org.apache.hadoop.ha.ActiveStandbyElector: Writing znode /hadoop-ha/jimmcluster/ActiveBreadCrumb to indicate that the local node is the most recent active...
2014-03-22 20:58:43,816 INFO org.apache.hadoop.ha.ZKFailoverController: Trying to make NameNode at nn/211.211.211.210:9000 active...
2014-03-22 20:58:45,645 INFO org.apache.hadoop.ha.ZKFailoverController: Successfully transitioned NameNode at nn/211.211.211.210:9000 to active state
```

第八步：启动其他所有服务

主要是启动 datanode：

```
start-dfs.sh
```

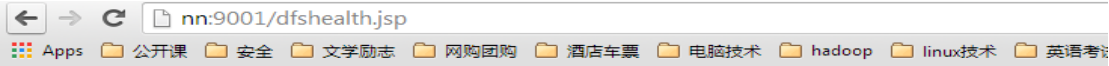
接下来测试是否添加成功：

```
[hadoop@nn hadoop]$ hdfs haadmin -getServiceState nn
14/03/21 18:01:09 WARN util.NativeCodeLoader: Unable to load native-hadoop
ere applicable
active
[hadoop@nn hadoop]$ hdfs haadmin -getServiceState nn1
14/03/21 18:01:22 WARN util.NativeCodeLoader: Unable to load native-hadoop
ere applicable
standby
```

执行：hdfs dfs -ls /app 查看原来的数据是否存在

```
[hadoop@nn hadoop]$ hdfs dfs -ls /app
14/03/21 18:01:57 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java versions where applicable
Found 32 items
drwxr-xr-x - hadoop hadoop 0 2014-03-21 13:25 /app/bak.xml
-rw-r--r-- 3 hadoop hadoop 3560 2014-03-21 13:25 /app/capacity-scheduler.xml
drwxr-xr-x - hadoop hadoop 0 2014-03-21 13:25 /app/cluster3n
-rw-r--r-- 3 hadoop hadoop 1335 2014-03-21 13:25 /app/configuration.xml
-rw-r--r-- 3 hadoop hadoop 318 2014-03-21 13:25 /app/container-executor.cfg
-rw-r--r-- 3 hadoop hadoop 2781 2014-03-21 13:25 /app/core-site.xml
-rw-r--r-- 3 hadoop hadoop 3589 2014-03-21 13:25 /app/hadoop-env.cmd
-rw-r--r-- 3 hadoop hadoop 3400 2014-03-21 13:25 /app/hadoop-env.sh
-rw-r--r-- 3 hadoop hadoop 2490 2014-03-21 13:25 /app/hadoop-metrics.properties
-rw-r--r-- 3 hadoop hadoop 1774 2014-03-21 13:25 /app/hadoop-metrics2.properties
-rw-r--r-- 3 hadoop hadoop 8257 2014-03-21 13:25 /app/hadoop-policy.xml
```

通过页面查看：

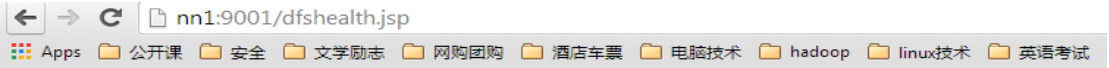


NameNode 'nn:9000' (active)

Started:	Fri Mar 21 14:57:15 CST 2014
Version:	2.2.0, 1529768
Compiled:	2013-10-07T06:28Z by hortonmu from branch-2.2.0
Cluster ID:	CID-2db5ba3b-56e9-498e-9111-79cf630e847c
Block Pool ID:	BP-1385802010-211.211.211.210-1395379438127

[Browse the filesystem](#)
[NameNode Logs](#)

Cluster Summary



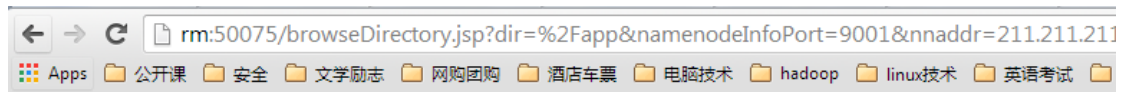
NameNode 'nn1:9000' (standby)

Started:	Fri Mar 21 14:57:24 CST 2014
Version:	2.2.0, 1529768
Compiled:	2013-10-07T06:28Z by hortonmu from branch-2.2.0
Cluster ID:	CID-2db5ba3b-56e9-498e-9111-79cf630e847c
Block Pool ID:	BP-1385802010-211.211.211.210-1395379438127

[NameNode Logs](#)

Cluster Summary

浏览文件系统文件：



Contents of directory /app

Goto :

[Go to parent directory](#)

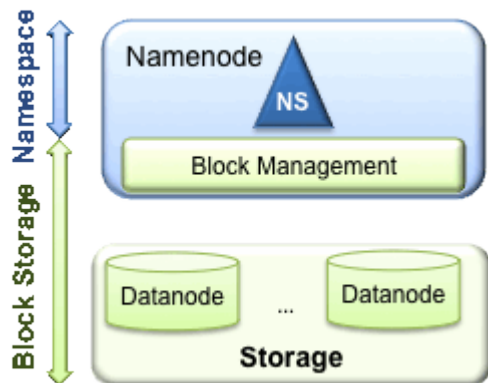
Name	Type	Size	Replication	Block Size	Modification Time	Permission	Owner	Group
bak.xml	dir				2014-03-21 13:25	rw-r--r--	hadoop	hadoop
capacity-scheduler.xml	file	3.48 KB	3	128 MB	2014-03-21 13:25	rw-r--r--	hadoop	hadoop
cluster3n	dir				2014-03-21 13:25	rw-r--r--	hadoop	hadoop
configuration.xml	file	1.30 KB	3	128 MB	2014-03-21 13:25	rw-r--r--	hadoop	hadoop
container-executor.cfg	file	318 B	3	128 MB	2014-03-21 13:25	rw-r--r--	hadoop	hadoop
core-site.xml	file	2.72 KB	3	128 MB	2014-03-21 13:25	rw-r--r--	hadoop	hadoop
hadoop-env.xml	file	3.50 KB	3	128 MB	2014-03-21 13:25	rw-r--r--	hadoop	hadoop

可以停止 active 来做测试，看是否会自动切换到另一台

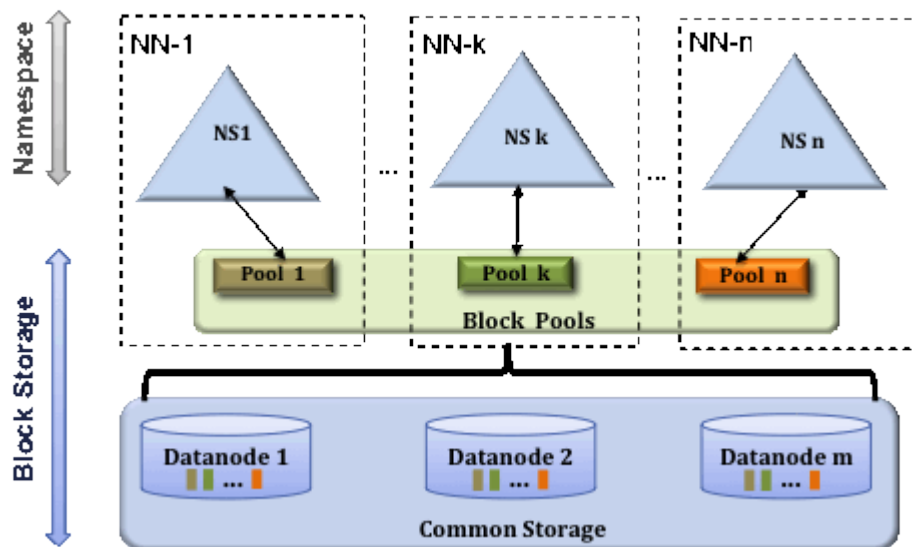
Federation

参考：<http://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-hdfs/Federation.html>

hdfs1.0



hdfs federation



暂时原理性的东西理解不够，需要加强理解，最好看看 hadoop 权威指南或其他相关书籍

开始进入 hbase 的学习

Hbase 安装于配置

参考：<http://hbase.apache.org/>
<http://abloz.com/hbase/book.html>

HBase 介绍

HBase - Hadoop Database，是一个高可靠性、高性能、面向列、可伸缩的分布式存储系统，利用 HBase 技术可在廉价 PC Server 上搭建起大规模结构化存储集群。

HBase 是 Google Bigtable 的开源实现，类似 Google Bigtable 利用 GFS 作为其

文件存储系统，HBase 利用 Hadoop HDFS 作为其文件存储系统；Google 运行 MapReduce 来处理 Bigtable 中的海量数据，HBase 同样利用 Hadoop MapReduce 来处理 HBase 中的海量数据；Google Bigtable 利用 Chubby 作为协同服务，HBase 利用 Zookeeper 作为对应。

摘自：<http://abloz.com/hbase/book.html>

搭建 HBase 单机环境

注意下载的版本一定要和 hadoop 版本对应才行，否则会启动不了，我这里用的 hadoop-2.2.0，可以去 lib 目录下看看是否有 hadoop-*-2.2.0.jar，如果和版本不对应，去官网重下，

```
hadoop-auth-2.2.0.jar
hadoop-client-2.2.0.jar
hadoop-common-2.2.0.jar
r hadoop-hdfs-2.2.0.jar
hadoop-hdfs-2.2.0-tests.jar
hadoop-mapreduce-client-app-2.2.0.jar
hadoop-mapreduce-client-common-2.2.0.jar
```

```
wget http://mirror.bit.edu.cn/apache/hbase/hbase-0.98.0/hbase-0.98.0-hadoop2-bin.tar.gz
```

```
tar -zxvf hbase-0.98.0-hadoop2-bin.tar.gz
cd hbase-0.98.0-hadoop2
```

修改/etc/profile

```
export JAVA_HOME=/usr/java/jdk1.6.0_38/
export HBASE_HOME=/home/app/hbase-0.94.17-security/
export HADOOP_HOME=/home/app/hadoop-2.2.0/
export ZOOKEEPER_HOME=/home/app/zookeeper-3.4.5/
export YARN_CONF_DIR=/home/app/hadoop-2.2.0/etc/hadoop/
export PATH=$JAVA_HOME/bin:$HADOOP_HOME/bin:
$HADOOP_HOME/sbin:/home/hadoop/bin:$ZOOKEEPER_HOME/bin:
$HBASE_HOME/bin:$PATH
```

修改 hbase-env.sh 中的 JAVA 的路径，就直接可以启动单机模式测试了

测试是否启动成功，摘自官网

执行：`hbase shell`

输入 `help` 然后 `<RETURN>` 可以看到一系列 shell 命令。这里的帮助很详细，要注意的是表名，行和列需要加引号。

创建一个名为 `test` 的表，这个表只有一个列族为 `cf`。可以列出所有的表来检

查创建情况，然后插入些值。

```
hbase(main):003:0> create 'test', 'cf'
0 row(s) in 1.2200 seconds
hbase(main):003:0> list 'table'
test
1 row(s) in 0.0550 seconds
hbase(main):004:0> put 'test', 'row1', 'cf:a', 'value1'
0 row(s) in 0.0560 seconds
hbase(main):005:0> put 'test', 'row2', 'cf:b', 'value2'
0 row(s) in 0.0370 seconds
hbase(main):006:0> put 'test', 'row3', 'cf:c', 'value3'
0 row(s) in 0.0450 seconds
```

以上我们分别插入了 3 行。第一个行 key 为 row1, 列为 cf:a, 值是 value1。HBase 中的列是由 列族前缀和列的名字组成的，以冒号间隔。例如这一行的列名就是 a。

检查插入情况。

Scan 这个表，操作如下

```
hbase(main):007:0> scan 'test'
ROW    COLUMN+CELL
row1    column=cf:a, timestamp=1288380727188, value=value1
row2    column=cf:b, timestamp=1288380738440, value=value2
row3    column=cf:c, timestamp=1288380747365, value=value3
3 row(s) in 0.0590 seconds
```

Get 一行，操作如下

```
hbase(main):008:0> get 'test', 'row1'
COLUMN    CELL
cf:a      timestamp=1288380727188, value=value1
1 row(s) in 0.0400 seconds
```

disable 再 drop 这张表，可以清除你刚刚的操作

```
hbase(main):012:0> disable 'test'
0 row(s) in 1.0930 seconds
hbase(main):013:0> drop 'test'
0 row(s) in 0.0770 seconds
```

关闭 shell

```
hbase(main):014:0> exit
```

停止集群：stop-hbase.sh

搭建 Hbase 分布式环境

#修改 hbase-env.sh

```
export JAVA_HOME=/usr/java/jdk1.6.0_38/  
export HBASE_PID_DIR=/home/dir/hbase pid 文件存放位置，也可默认  
export HBASE_MANAGES_ZK=false 开启使用外部 zookeeper 集群
```

#搭建 zookeeper 集群

参照前面 zookeeper 集群配置

修改 hbase-site.xml

```
<?xml version="1.0"?>  
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>  
<configuration>  
  <property>  
    <name>hbase.rootdir</name>  
    <value>hdfs://nn1:9000/hbase/data</value>  
    <description>Enter the HBase NameNode server hostname</description>  
  </property> 这里填写 HDFS 分布式文件系统的路径  
  <property>  
    <name>hbase.master.info.bindAddress</name>  
    <value>dn</value>  
    <description>Enter the HBase Master server hostname</description>  
  </property>  
  <property>  
    <name>hbase.zookeeper.quorum</name>  
    <value>rm,nn1,nn</value>  
    <description>Comma separated list of Zookeeper servers (match to what isspecified  
in zoo.cfg but without portnumbers)</description>  
  </property> zookeeper 集群的所以主机名  
  <property>  
    <name>hbase.cluster.distributed</name>  
    <value>true</value>  
  </property> 需要开启分布式模式，默认是未开启，导致无法启动 regionserves  
</configuration>
```

#修改 regionservers

这里存储了所有的 regionserver

```
[hadoop@dn conf]$ cat regionservers  
dn  
nn1
```

#启动 hbase 集群


start-hbase.sh

#查看

```
[hadoop@dn conf]$ start-hbase.sh
starting master, logging to /home/app/hbase-0.98.0-hadoop2//logs/hbase-hadoop-master-dn.out
nn1: starting regionserver, logging to /home/app/hbase-0.98.0-hadoop2/bin/../logs/hbase-hadoop-regionserver-nn1.out
dn: starting regionserver, logging to /home/app/hbase-0.98.0-hadoop2/bin/../logs/hbase-hadoop-regionserver-dn.out

[hadoop@dn conf]$ jps
16064 JournalNode
15995 DataNode
10851 Jps
10592 HRegionServer
10466 HMaster
```

页面查看 dn:60010

 Home Table Details Local logs Log Level Debug dump Metrics Dump HBase Configuration

Master dn

Region Servers

Base Stats Memory Requests Storefiles Compactions

ServerName	Start time	Requests Per Second	Num. Regions
dn,60020,1395824959913	Wed Mar 26 17:09:19 CST 2014	0	3
nn1,60020,1395824953495	Wed Mar 26 17:09:13 CST 2014	0	1
Total:2		0	4

OK，搭建完成

但是对 Hbase 原理了解的太少了，加强原理学习，hadoop 也是

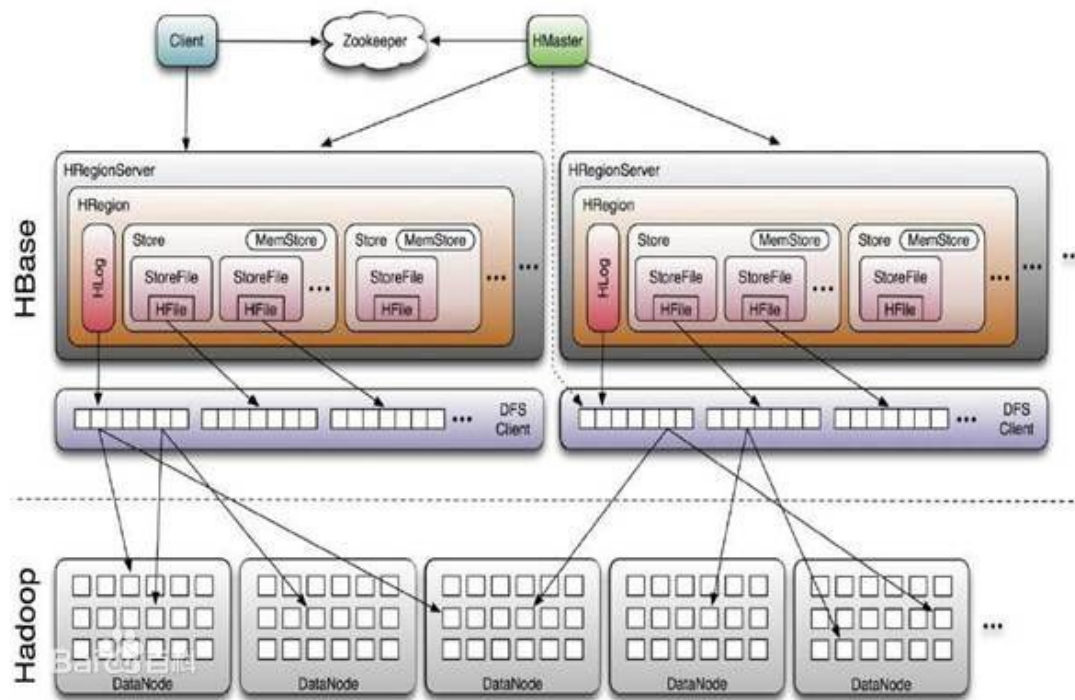
问题：

1，当 HDFS 为 HA 时，有两台 namenode，Hbase 的配置文件中配的是 active 那台 namenode，当 active 集群 down 掉时，会导致 Hbase 无法连接上 HDFS，怎么解决？

2,HDFS 在使用过程中两台 HA 会每隔一段时间互相切换，可以分析日志看看

3,该怎么深入学习 hadoop 生态系统

Hbase 原理简介



Hbase 配置文件

详见 <http://abloz.com/hbase/book.html#config.files>

hbase.rootdir

这个目录是 region server 的共享目录，用来持久化 HBase。URL 需要是'完全正确'的，还要包含文件系统的 scheme。例如，要表示 hdfs 中的 '/hbase' 目录，namenode 运行在 namenode.example.org 的 9090 端口。则需要设置为 hdfs://namenode.example.org:9000/hbase。默认情况下 HBase 是写到 /tmp 的。不改这个配置，数据会在重启的时候丢失。

默认: file:///tmp/hbase-\${user.name}/hbase

hbase.master.port

HBase 的 Master 的端口。

默认: 60000

hbase.cluster.distributed

HBase 的运行模式。false 是单机模式，true 是分布式模式。若为 false, HBase 和 Zookeeper 会运行在同一个 JVM 里面。

默认: false

hbase.tmp.dir

本地文件系统的临时文件夹。可以修改到一个更为持久的目录上。(/tmp 会在重启时清楚)

默认: \${java.io.tmpdir}/hbase-\${user.name}

hbase.local.dir

作为本地存储，位于本地文件系统的路径。

默认: \${hbase.tmp.dir}/local/

hbase.master.info.port

HBase Master web 界面端口. 设置为-1 意味着你不想让他运行。

默认: 60010

hbase.master.info.bindAddress

HBase Master web 界面绑定的端口

默认: 0.0.0.0

hbase.client.write.buffer

HTable 客户端的写缓冲的默认大小。这个值越大，需要消耗的内存越大。因为缓冲在客户端和服务端都有实例，所以需要消耗客户端和服务端两个地方的内存。得到的好处是，可以减少 RPC 的次数。可以这样估算服务器端被占用的内存： $\text{hbase.client.write.buffer} * \text{hbase.regionserver.handler.count}$

默认: 2097152

hbase.regionserver.port

HBase RegionServer 绑定的端口

默认: 60020

hbase.regionserver.info.port

HBase RegionServer web 界面绑定的端口 设置为 -1 意味这你不想与运行

RegionServer 界面.

默认: 60030

hbase.regionserver.info.port.auto

Master 或 RegionServer 是否要动态搜一个可以用的端口来绑定界面。当 hbase.regionserver.info.port 已经被占用的时候，可以搜一个空闲的端口绑定。这个功能在测试的时候很有用。默认关闭。

默认: false

hbase.regionserver.info.bindAddress

HBase RegionServer web 界面的 IP 地址

默认: 0.0.0.0

Hbase 数据模型

简单来说，应用程序是以表的方式在 HBase 存储数据的。表是由行和列构成的，所有的列是从属于某一个列族的。行和列的交叉点称之为 cell, cell 是版本化的。cell 的内容是不可分割的字节数组。

表的行键也是一段字节数组，所以任何东西都可以保存进去，不论是字符串或者数字。HBase 的表是按 key 排序的，排序方式之针对字节的。所有的表都必须要有主键-key。

貌似对数据库这一刻不怎么熟悉，或许可以抽个时间好好学一下，特别是 mysql，毕竟用的人太多了，另外 Hbase 还是可以再深入学习一下，或许有好处，下面就先了解一下 hbase 的命令吧。

Hbase 命令

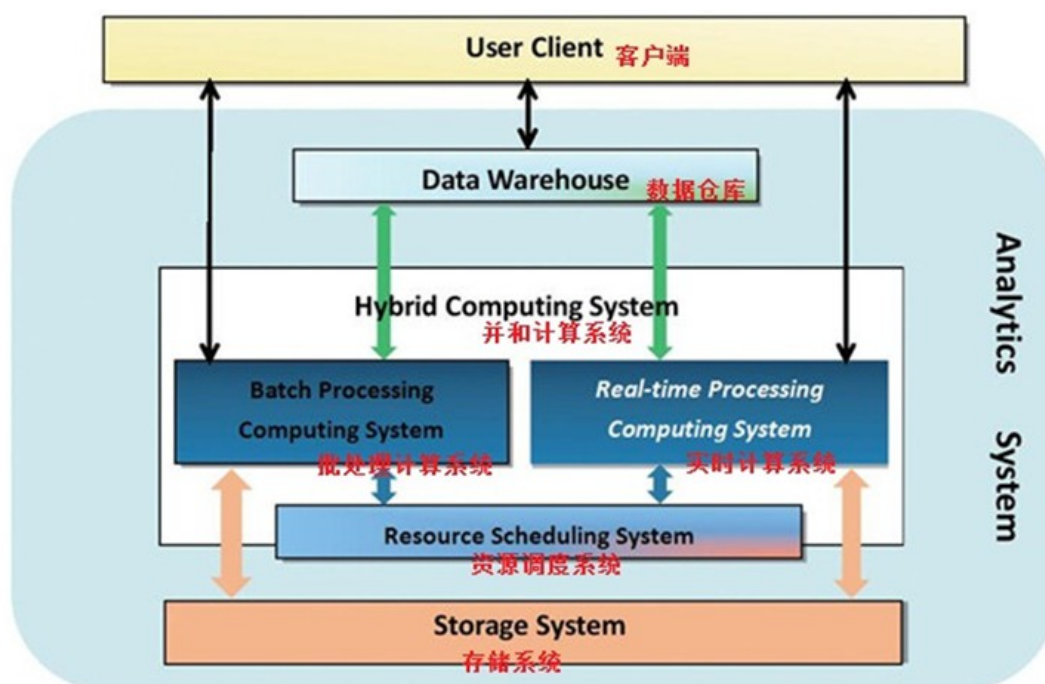
Storm 原理简介

1.1 实时流计算背景

随着互联网的更进一步发展，信息浏览、搜索、关系交互传递型，以及电子商务、互联网旅游生活产品等将生活中的流通环节在线化。对于实时性的要求进一步提升，而信息的交互和沟通正在从点对点信息链甚至信息网的方向发展，这样必然带来数据在各个维度的交叉关联，数据爆炸已不可避免。因此流式处理和 NoSQL 产品应运而生，分别解决实时框架和数据大规模存储计算的问题。

流式处理可以用于 3 种不同场景：事件流、持续计算以及分布式 RPC。

1.2 数据分析系统组成



1.3 流处理与批处理

Storm 关注的是数据多次处理一次写入，而 hadoop 关注的是数据一次写入，多次查询使用。Storm 系统运行起来后是持续不断的，而 hadoop 往往只是在业务需要时调用数据。

1.4 Storm 和 Hadoop 角色对比

	Hadoop	Storm
系统角色	JobTracker	Nimbus
	TaskTracker	Supervisor
	Child	Worker
应用名称	Job	Topology
组件接口	Mapper/Reducer	Spout/Bolt

1.5 Storm 组件

Topology：storm 中运行的一个实时应用程序。

Nimbus：负责资源分配和任务调度。

Supervisor：负责接受 nimbus 分配的任务,启动和停止属于自己管理的 worker 进程。

Worker：运行具体处理组件逻辑的进程。

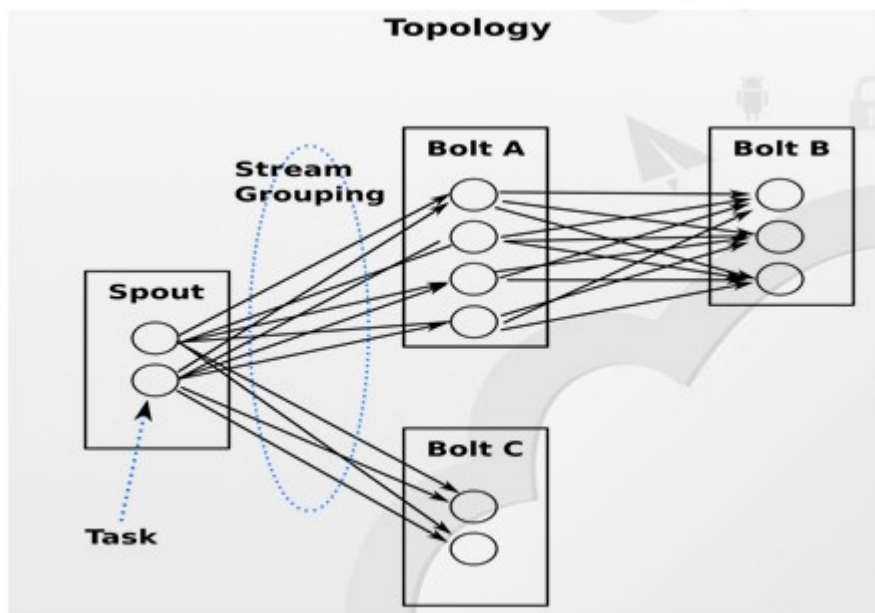
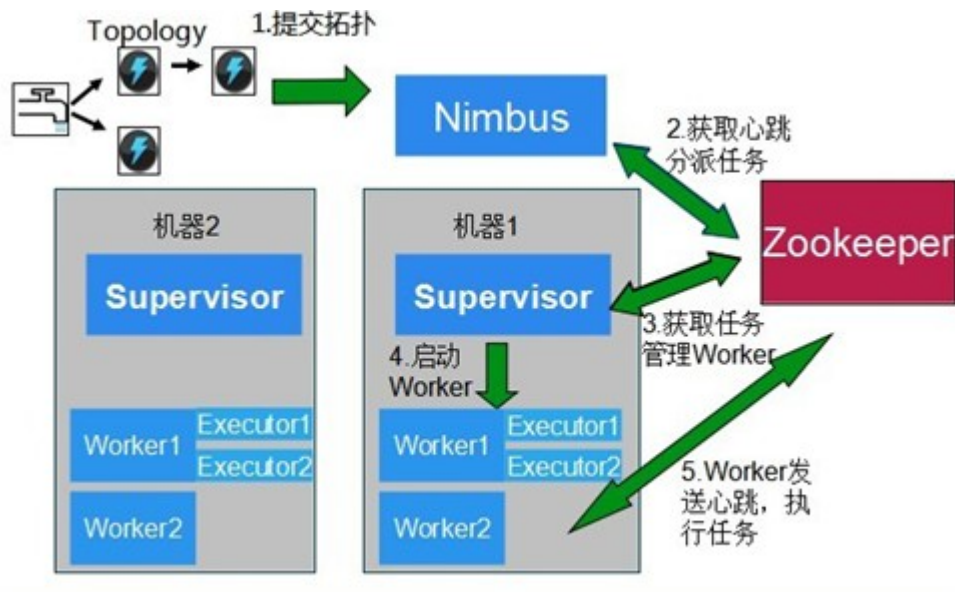
Task：worker 中每一个 spout/bolt 的线程称为一个 task。

Spout：在一个 topology 中产生源数据流的组件。

Bolt：在一个 topology 中接受数据然后执行处理的组件。

Tuple：一次消息传递的基本单元。

Stream grouping：消息的分组方法



1.6 Storm 特点

□ 可扩展

计算任务可在多个线程、进程和服务端之间并行进行,支持灵活的水平扩展

□ 高可靠

保证每条消息都能被完全处理

□ 高容错性

nimbus、supervisor 都是无状态的,可以用 kill -9 来杀死 Nimbus 和 Supervisor 进程,然后再重启它们,任务照常进行.当 worker 失败后,supervisor 会尝试在本机重启它

□ 支持多种编程语言

除了用 java 实现 spout 和 bolt,还可用其他语言

□ 支持本地模式

可在本地模拟一个 Storm 集群功能、进行本地测试

□ 高效

用 ZeroMQ 作为底层消息队列, 保证消息能快速被处理

Storm 安装配置

1, 需要在 Nimbus 和 Supervisor 机器上安装 Storm 的依赖库, 具体如下:

1. zeromq-4.0.4 (<http://download.zeromq.org/zeromq-4.0.4.tar.gz>)
2. JZMQ (git clone <https://github.com/nathanmarz/jzmq.git>)
3. Libtool (JZMQ 编译依赖库)
4. Java 6
5. Python 2.6.6
6. Unzip

同时 nimbus 与 supervisor 通信需要依赖 zookeeper 集群, 所以需要搭建 zookeeper 集群, 参照前面 zookeeper 集群搭建。

To install ZeroMQ, run:

```
wget http://download.zeromq.org/zeromq-4.0.4.tar.gz
tar -xzf zeromq-4.0.4.tar.gz
cd zeromq-4.0.4
```

```
./configure
```

```
make
```

```
sudo make install
```

To install JZMQ, run:

```
git clone https://github.com/nathanmarz/jzmq.git
```

```
cd jzmq
```

```
./autogen.sh
```

```
./configure
```

```
make
```

```
sudo make install
```

其他自行安装,

2, 修改 storm.yaml 配置文件

storm.zookeeper.servers:

- "nn1"
- "nn2"
- "rm"

nimbus.host: "hadoop"

storm.zookeeper.servers

The address of your zookeeper servers.

nimbus.host

The address of the machine where the Nimbus will run.

3, 启动

以下是启动 Storm 各个后台进程的方式：

1. **Nimbus:** 在 Storm 主控节点上运行"`bin/storm nimbus >/dev/null 2>&1 &`"启动 Nimbus 后台程序，并放到后台执行；
2. **Supervisor:** 在 Storm 各个工作节点上运行"`bin/storm supervisor >/dev/null 2>&1 &`"启动 Supervisor 后台程序，并放到后台执行；
3. **UI:** 在 Storm 主控节点上运行"`bin/storm ui >/dev/null 2>&1 &`"启动 UI 后台程序，并放到后台执行，启动后可以通过 `http://{nimbus host}:8080` 观察集群的 worker 资源使用情况、Topologies 的运行状态等信息。

5, 向集群提交任务

启动 Storm Topology：

`storm jar allmycode.jar org.me.MyTopology arg1 arg2 arg3`

其中，allmycode.jar 是包含 Topology 实现代码的 jar 包，org.me.MyTopology 的 main 方法是 Topology 的入口，arg1、arg2 和 arg3 为 org.me.MyTopology 执行时需要传入的参数。

停止 Storm Topology：

`storm kill {toponame}`

其中，{toponame}为 Topology 提交到 Storm 集群时指定的 Topology 任务名称。

Flume 分布式日志采集系统

原理介绍

官网：<http://flume.apache.org>

用户手册：<http://flume.apache.org/FlumeUserGuide.html>

- flume 是什么

Apache Flume is a distributed, reliable, and available system for efficiently collecting, aggregating and moving large amounts of log data from many different sources to a

Flume 是一个从不同节点采集元数据到集中存储中心中心的分布式、可靠、和高可用的海量日志采集、聚合和传输系统。

支持在系统中定制各类数据发送方，用于收集数据；同时，Flume 提供对数据进行简单处理，并写到各种数据接受方（可定制）的能力。

Flume 作为 cloudera 开发的实时日志收集系统，已经受到越来越多的关注。比如 IBM BigInsights 已经将 Flume 作为产品的一部分。Flume 初始的发行版本目前被统称为 Flume OG (original generation)，属于 cloudera。但随着 Flume 功能的扩展，Flume OG 代码工程臃肿、核心组件设计不合理、核心配置不标准等缺点暴露出来，尤其是在 Flume OG 的最后一个发行版本 0.94.0 中，日志传输不稳定的现象尤为严重，这点可以在 BigInsights 产品文档的 troubleshooting 板块发现。为了解决这些问题，2011 年 10 月 22 号，cloudera 完成了 Flume-728，对 Flume 进行了里程碑式的改动：重构核心组件、核心配置以及代码架构，重构后的版本统称为 Flume NG (next generation)；改动的另一原因是将 Flume 纳入 apache 旗下，cloudera Flume 改名为 Apache Flume。

IBM 的这篇文章：[Flume NG：Flume 发展史上的第一次革命](#)，从基本组件以及用户体验的角度阐述 Flume OG 到 Flume NG 发生的革命性变化。

● flume 核心概念

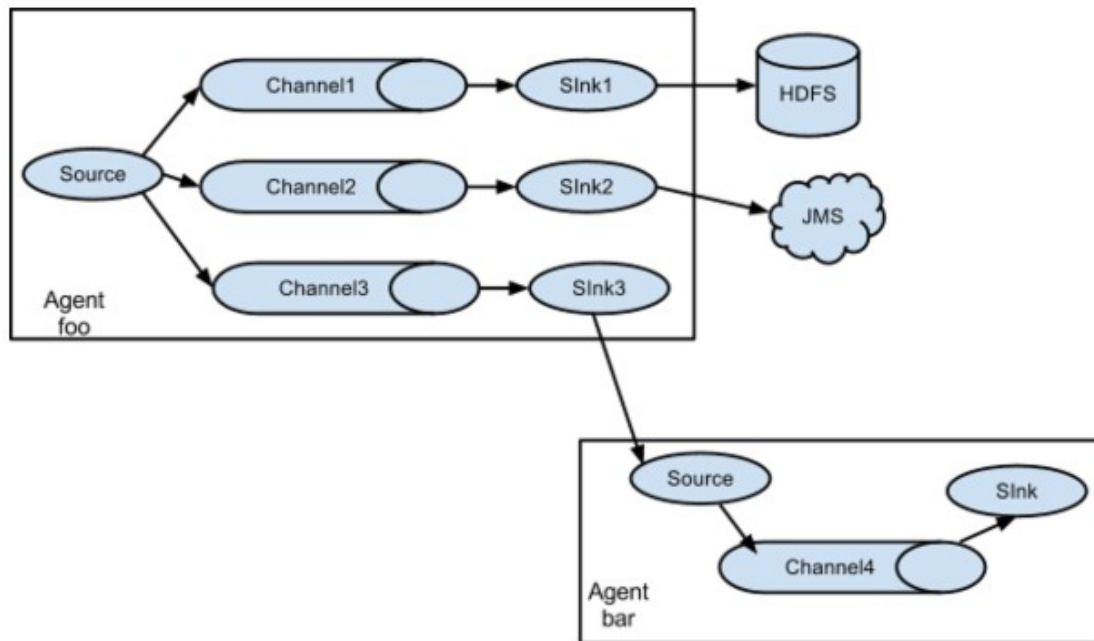
Flume NG 中的核心概念：

- Client：生产数据，运行在一个独立的线程。
- Source：从 Client 收集数据，传递给 Channel。可以接收外部源发送过来的数据。不同的 source，可以接受不同的数据格式。比如有目录池(spooling directory)数据源，可以监控指定文件夹中的新文件变化，如果目录中有文件产生，就会立刻读取其内容。
- Channel：是一个存储地，接收 source 的输出，直到有 sink 消费掉 channel 中的数据。Channel 中的数据直到进入到下一个 channel 中或者进入终端才会被删除。当 sink 写入失败后，可以自动重启，不会造成数据丢失，因此很可靠。
- Sink：会消费 channel 中的数据，然后送给外部源或者其他 source。如数据可以写入到 HDFS 或者 HBase 中。
- Agent：使用 JVM 运行 Flume。每台机器运行一个 agent，但是可以在一个 agent 中包含多个 sources 和 sinks。
- Events：Flume NG 传输的数据的基本单位是 event，如果是文本文件，通常是一行记录，这也是事务的基本单位。

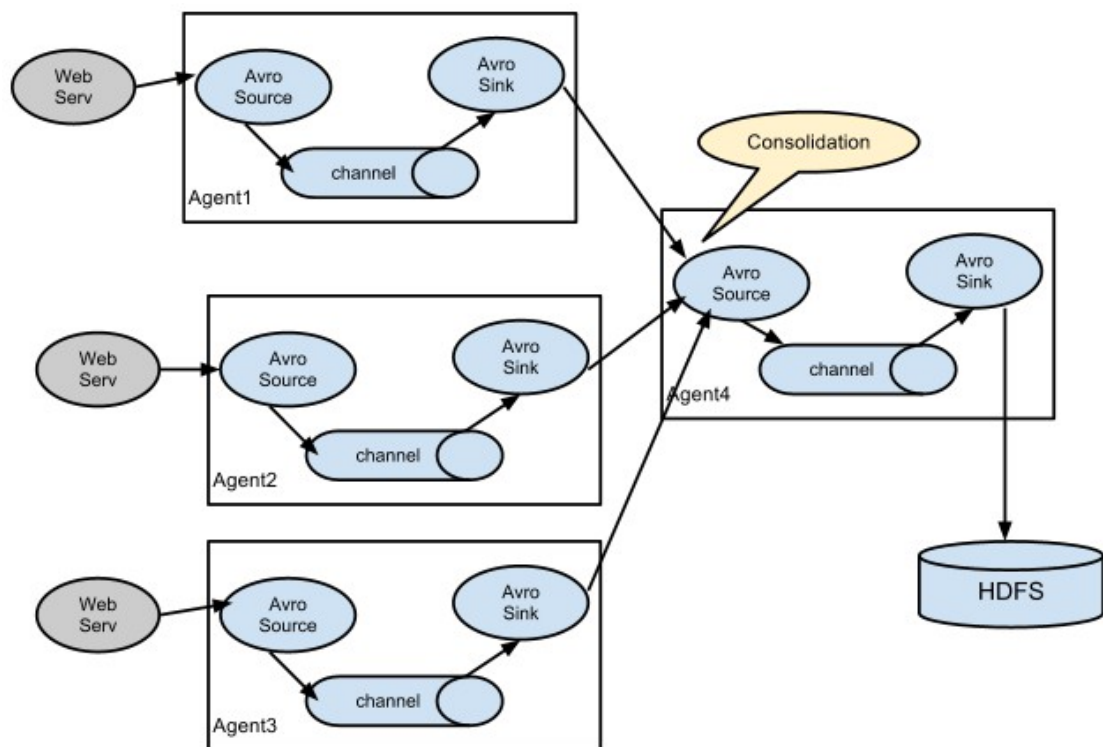
架构理解

Flume NG 以 agent 为最小的独立运行单位。一个 agent 就是一个 JVM。单 agent 由 Source、Sink 和 Channel 三大组件构成。

Flume 的数据流由事件(Event)贯穿始终。事件是 Flume 的基本数据单位，它携带日志数据(字节数组形式)并且携带有头信息，这些 Event 由 Agent 外部的 Source，比如上图中的 Web Server 生成。当 Source 捕获事件后会进行特定的格式化，然后 Source 会把事件推入(单个或多个)Channel 中。可以把 Channel 看作是一个缓冲区，它将保存事件直到 Sink 处理完该事件。Sink 负责持久化日志或者把事件推向另一个 Source。值得注意的是，Flume 提供了大量内置的 Source、Channel 和 Sink 类型。不同类型的 Source、Channel 和 Sink 可以自由组合。组合方式基于用户设置的配置文件，非常灵活。比如：Channel 可以把事件暂存在内存里，也可以持久化到本地硬盘上。Sink 可以把日志写入 HDFS, HBase，甚至是另外一个 Source 等等。Flume 支持用户建立多级流，也就是说，多个 agent 可以协同工作，并且支持 Fan-in、Fan-out、Contextual Routing、Backup Routes。如下图：



Flume 允许多个 agent 连在一起，形成前后相连的多级跳：



各组件原理介绍

● source

Flume 支持 Avro, log4j, syslog 和 http post(body 为 json 格式)。可以让应用程序同已有的 Source 直接打交道, 如 AvroSource, SyslogTcpSource。也可以写一个 Source, 以 IPC 或 RPC 的方式接入自己的应用, Avro 和 Thrift 都可以(分别有 NettyAvroRpcClient 和 ThriftRpcClient 实现了 RpcClient 接口), 其中 Avro 是默认的 RPC 协议。具体代码级别的 Client 端数据接入, 可以参考官方手册。对现有程序改动最小的使用方式是使用是直接读取程序原来记录的日志文件, 基本可以实现无缝接入, 不需要对现有程序进行任何改动。对于直接读取文件 Source, 有两种方式:

- ExecSource: 以运行 Linux 命令的方式, 持续的输出最新的数据, 如 tail -F 文件名 指令, 在这种方式下, 取的文件名必须是指定的。ExecSource 可以实现对日志的实时收集, 但是存在 Flume 不运行或者指令执行出错时, 将无法收集到日志数据, 无法保证日志数据的完整性。
- SpoolSource: 监测配置的目录下新增的文件, 并将文件中的数据读取出来。需要注意两点: 拷贝到 spool 目录下的文件不可以再打开编辑; spool 目录下不可包含相应的子目录。SpoolSource 虽然无法实现实时的收集数据, 但是可以使用以分钟的方式分割文件, 趋近于实时。如果应用无法实现以分钟切割日志文件的话, 可以两种收集方式结合使用。在实际使用的过程中, 可以结合 log4j 使用, 使用 log4j 的时候, 将 log4j 的文件分割机制设为 1 分钟一次, 将文件拷贝到 spool 的监控目录。log4j 有一个 TimeRolling 的插件, 可以把 log4j 分割文件到 spool 目录。基本实现了实时的监控。Flume 在传完文件之后, 将会修改文件的后缀, 变为 .COMPLETED (后缀也可以在配置文件中灵活指定)

● Channel

当前有几个 channel 可供选择, 分别是 Memory Channel, JDBC Channel, File Channel, Psuedo Transaction Channel。比较常见的是前三种 channel。

- MemoryChannel 可以实现高速的吞吐, 但是无法保证数据的完整性。
- MemoryRecoverChannel 在官方文档的建议上已经建议使用 FileChannel 来替换。
- FileChannel 保证数据的完整性与一致性。在具体配置 FileChannel 时, 建议 FileChannel 设置的目录和程序日志文件保存的目录设成不同的磁盘, 以便提高效率。

File Channel 是一个持久化的隧道 (channel), 它持久化所有的事件, 并将其存储到磁盘中。因此, 即使 Java 虚拟机当掉, 或者操作系统崩溃或重启, 又或者事件没有在管道中成功地传递到下一个代理 (agent), 这一切都不会造成数据丢失。Memory Channel 是一个不稳定的隧道, 其原因是由于它在内存中存储所有事件。如果 java 进程死掉, 任何存储在内存的事件将会丢失。另外, 内存的空间收到 RAM 大小的限制, 而 File Channel 这方面是它的优势, 只要磁盘空间足够, 它就可以将所有事件数据存储到磁盘上。

● sink

Sink 在设置存储数据时，可以向文件系统、数据库、hadoop 存数据，在日志数据较少时，可以将数据存储在文件系统中，并且设定一定的时间间隔保存数据。在日志数据较多时，可以将相应的日志数据存储到 Hadoop 中，便于日后进行相应的数据分析。更多 sink 的内容可以参考[官方手册](#)。

从整体上讲，NG 在核心组件上进行了大规模的调整，核心组件的数目由 7 删减到 4。由于 Flume 的使用涉及到众多因素，如 avro、thrift、hdfs、jdbc、zookeeper 等，而这些组件和 Flume 的整合都需要关联到所有组件。所以核心组件的改革对整个 Flume 的使用影响深远：

- 大大降低了对用户的要求，如不再依赖 zookeeper，用户无需去搭建 zookeeper 集群
- 用户也不再纠结于 OG 中的模糊概念（尤其是 physical nodes、logical nodes、agent、collector）
- 有利于 Flume 和其他技术、hadoop 周边组件的整合，比如在 NG 版本中，Flume 轻松实现了和 jdbc、hbase 的集成
- 将 OG 版本中复杂、大规模、不稳定的标签移除，Flume 实现了向灵活、轻便的转变，而且在功能上更加强大、可扩展性更高

参考链接：

- Flume 官方网站：<http://flume.apache.org/>
- Flume 用户文档：<http://flume.apache.org/FlumeUserGuide.html>
- Flume 开发文档：<http://flume.apache.org/FlumeDeveloperGuide.html>
- Scribe、Chukwa、Kafka、Flume 日志系统对比：<http://www.ttlsa.com/log-system/scribe-chukwa-kafka-flume-log-system-contrast/>
- 关于 Flume-ng 那些事：<http://www.ttlsa.com/?s=flume>
- Flume 1.4.0 User Guide：<http://archive.cloudera.com/cdh4/cdh/4/flume-ng-1.4.0-cdh4.7.0/FlumeUserGuide.html>
- Flume 日志采集：http://blog.csdn.net/sunmeng_007/article/details/9762507
- Flume-NG + HDFS + HIVE 日志收集分析：
<http://eyelublog.wordpress.com/2013/01/13/flume-ng-hdfs-hive-%E6%97%A5%E5%BF%97%E6%94%B6%E9%9B%86%E5%88%86%E6%9E%90/>
- Flume-ng+Kafka+Storm+HDFS 实时系统搭建：
<http://blog.csdn.net/weijonathan/article/details/18301321>
- Flume-NG + HDFS + PIG 日志收集分析：
http://hi.baidu.com/life_to_you/item/a98e2ec3367486dbef183b5e
- Flume 示例一收集 tomcat 日志：<http://my.oschina.net/88sys/blog/71529>
- Flume-ng 多节点集群示例：<http://my.oschina.net/u/1401580/blog/204052>
- 试用 Flume-ng 1.1：<http://heipark.iteye.com/blog/1617995>

- 基于 Flume 的美团日志收集系统(一)架构和设计：<http://tech.meituan.com/mt-log-system-arch.html>
- 基于 Flume 的美团日志收集系统(二)改进和优化：<http://tech.meituan.com/mt-log-system-optimization.html>
- 美团对 flume 的改进代码：<https://github.com/dashengju/mt-flume>

部署配置

● 安装

安装很简单，直接下载解压就能使用了

<http://www.apache.org/dyn/closer.cgi/flume/1.5.2/apache-flume-1.5.2-bin.tar.gz>

● 配置文件

配置文件在 conf 目录下面，首先配置 `flume-env.sh` 中的 jdk 路径

```
JAVA_HOME=/usr/java/jdk1.6.0_38/
```

接着配置 `flume-conf.properties` 启动需要的文件，这里列举三类示例

■ 单机测试配置

例一：

```
$cat seq.properties
```

```
#list agent
```

```
nn1.sources = sequence
```

```
nn1.channels = mm
```

```
nn1.sinks = logger
```

```
#source
```

```
nn1.sources.sequence.type = seq
```

```
nn1.sources.sequence.channels = mm
```

```
#channel
```

```
nn1.channels.mm.type = memory
```

```
nn1.channels.mm.capacity = 10000
```

```
nn1.channels.mm.transactionCapacity = 1000
```

```
#sink
```

```
nn1.sinks.logger.type = logger
```

```
nn1.sinks.logger.channel = mm
```

```
执行启动：../bin/flume-ng agent --f seq.properties --name  
nn1 -Dflume.root.logger=INFO,console
```

这会持续输出序列数值

例二：

```
$cat hello.properties
```

```
#list agent
```

```
nn1.sources = hello
```

```
nn1.channels = mm
```

```
nn1.sinks = logger
```

```
#source
```

```
nn1.sources.hello.type = exec
```

```
nn1.sources.hello.shell = /bin/bash -c
```

```
nn1.sources.hello.command = for i in `seq 100`;do echo "hello,flume";done
```

```
nn1.sources.hello.channels = mm
```

```
#channel
```

```
nn1.channels.mm.type = memory
```

```
nn1.channels.mm.capacity = 1000
```

```
nn1.channels.mm.transactionCapacity = 100
```

```
#sink
```

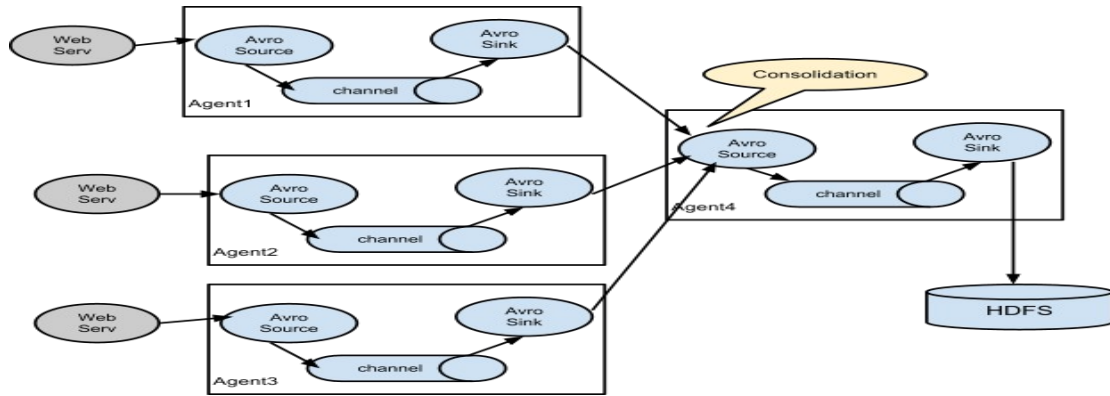
```
nn1.sinks.logger.type = logger
```

```
nn1.sinks.logger.channel = mm
```

```
执 行      :  $../bin/flume-ng      agent      --f      hello.properties      --name      nn1  
-Dflume.root.logger=INFO,console
```

这回输出 100 次 `hello,flume`

■ 输出到 HDFS 配置



这是一个典型的 C/S 结构，client 端将日志汇聚到 server 端，由 server 将日志写入 HDFS 集群，下面看配置

■ Client 端配置（所有 Client 端配置都一样）

```
$cat hdfs-conf.properties
```

```
#list agent
```

```
client-agent.sources = weblog
```

```
client-agent.channels = hdfs-channel
```

```
client-agent.sinks = hdfs-jimmycluster
```

```
#sources
```

```
client-agent.sources.weblog.type = exec
```

```
client-agent.sources.weblog.command = sudo tail -F /var/log/messages
```

```
client-agent.sources.weblog.channels = hdfs-channel
```

```
#channel
```

```
client-agent.channels.hdfs-channel.type = memory
```

```
client-agent.channels.hdfs-channel.capacity = 1000
```

```
client-agent.channels.hdfs-channel.transactioncapacity = 100
```

```
#sink-to-nn1
```

```
client-agent.sinks.hdfs-jimmycluster.type = avro
```

```
client-agent.sinks.hdfs-jimmycluster.channel = hdfs-channel
```

```
client-agent.sinks.hdfs-jimmycluster.hostname = nn1
```

```
client-agent.sinks.hdfs-jimmycluster.port = 41415
```

```
#client-agent.sinks.hdfs-jimmycluster.hdfs.filePrefix = apache-
```

```
#client-agent.sinks.hdfs-jimmycluster.hdfs.writeFormat = Text
```

```
#client-agent.sinks.hdfs-jimmycluster.hdfs.fileType = DataStream
```

```
#client-agent.sinks.hdfs-jimmycluster.hdfs.rollInterval = 0
```

```
#client-agent.sinks.hdfs-jimmycluster.hdfs.rollSize = 1000000
```

```
#client-agent.sinks.hdfs-jimmycluster.hdfs.rollCount = 0
```

```
#client-agent.sinks.hdfs-jimmycluster.hdfs.batchSize = 1000
```

```
#client-agent.sinks.hdfs-jimmycluster.hdfs.txnEventMax = 1000
```

```
#client-agent.sinks.hdfs-jimmycluster.hdfs.callTimeout = 60000
#client-agent.sinks.hdfs-jimmycluster.hdfs.appendTimeout = 60000
```

■ Server 端的配置

```
$cat hdfs-conf.properties
#list agent
hdfs-agent.sources = weblog
hdfs-agent.channels = hdfs-jimmy
hdfs-agent.sinks = hdfs-jimmycluster

#sources
hdfs-agent.sources.weblog.type = avro
hdfs-agent.sources.weblog.bind = nn1
hdfs-agent.sources.weblog.port = 41415
hdfs-agent.sources.weblog.channels = hdfs-jimmy

#channel
hdfs-agent.channels.hdfs-jimmy.type = memory
hdfs-agent.channels.hdfs-jimmy.capacity = 10000
hdfs-agent.channels.hdfs-jimmy.transactioncapacity = 1000

#sink-to-nn1
hdfs-agent.sinks.hdfs-jimmycluster.type = hdfs
hdfs-agent.sinks.hdfs-jimmycluster.channel = hdfs-jimmy
hdfs-agent.sinks.hdfs-jimmycluster.hdfs.path = hdfs://nn1:9000/http/
#hdfs-agent.sinks.hdfs-jimmycluster.hostname = nn1
#hdfs-agent.sinks.hdfs-jimmycluster.port = 41415
hdfs-agent.sinks.hdfs-jimmycluster.hdfs.filePrefix = apache-
hdfs-agent.sinks.hdfs-jimmycluster.hdfs.writeFormat = Text
hdfs-agent.sinks.hdfs-jimmycluster.hdfs.fileType = DataStream
hdfs-agent.sinks.hdfs-jimmycluster.hdfs.rollInterval = 0
hdfs-agent.sinks.hdfs-jimmycluster.hdfs.rollSize = 1000000
hdfs-agent.sinks.hdfs-jimmycluster.hdfs.rollCount = 0
hdfs-agent.sinks.hdfs-jimmycluster.hdfs.batchSize = 1000
hdfs-agent.sinks.hdfs-jimmycluster.hdfs.txnEventMax = 1000
hdfs-agent.sinks.hdfs-jimmycluster.hdfs.callTimeout = 60000
hdfs-agent.sinks.hdfs-jimmycluster.hdfs.appendTimeout = 60000
```

我这里为了测试简便采用了 memory 的 channel，这样机器 down 掉后会丢失数据，一般生产环境采用 channel 采用 file，结合 source 采用 spooling，当然也可以配置高可用及负载均衡，这个还没研究，后续补上

下面是 ibm 官网转载的

Properties Name	Description
agent_ff	用来收集日志信息的 agent 节点名称
agent_ff.source	需要收集的信息源，名字：tailsource-ff
agenter_ff.sinks	日志需要被收集到此处，名字：hdfsSink-ff。
agent_ff.channels	日志的收集需要通过此管道，名字：memoryChannel-ff。
tailsource-ff.type	定义 source 的类型，此处 exec 代表数据源是 exec 命令
tailsource-ff.command	定义具体命令，此处是对文件/var/log/secure 做 tail
tailsource-ff.channels	数据传输的管道，此处的管道名称应该和 sink 相同。从而将 source、sink 通过 channels 进行连接。
memoryChannel-ff.type	管道类型，代表事件存储的方式。Source 产生事件，sink 移除事件，以此代表数据传输完成。目前 Flume 支持 6 种 channel。此处是 momery，代表事件是存在内存里
memoryChannel-ff.capacity	管道里可以存放的最多的事件数目。此处代表 memoryChannel-ff 最多可存放 1000 个事件。
hdfsSink-ff.type	数据目的地的类型，此处是将数据存放在 hdfs 中。
hdfsSink-ff.channel	定义和 source 相关联的管道。
hdfsSink-ff.hdfs.path	数据存放在 hdfs 中的位置。
hdfsSink-ff.hdfs.filePrefix	收集到的数据存放的文件以此为前缀。如图 8。
hdfsSink-ff.hdfs.round, hdfsSink-ff.hdfs.roundValue, hdfsSink-ff.hdfs.roundUnit	定义在 hdfs 中生成的文件的时间戳。此处代表将 hdfs 中的文件的时间戳，向下取整到上一个十分钟内。比如说，在 2012 年 6 月 12 号上午 11:54:34 生成的事件，在 hdfs 中生成的路径将是/flume/events/2012-06-12/1150/00。

■ 负载均衡及高可用的相关配置

略

开发相关

