

Spark

运维实战

taoistwar

Published
with GitBook



目錄

Introduction	0
Spark 概述	1
Spark 生态环境	1.1
Spark 安装配置	2
Spark 编译	2.1
Spark 部署模式	2.2
Spark 单机版	2.2.1
Spark Standalone	2.2.2
无HA	2.2.2.1
单点恢复	2.2.2.2
ZooKeeper HA	2.2.2.3
配置项	2.2.2.4
Spark on Yarn	2.2.3
Spark Shell	2.3
Spark Submit	2.4
Spark Relate Software	3
ZooKeeper	3.1
ZooKeeper安装配置	3.1.1
Hadoop	3.2
Hadoop 1.x 安装配置	3.2.1
Hadoop 2.x 安装配置	3.2.2
kafka	3.3
kafka 安装配置	3.3.1
kafka 配置项	3.3.2
kafka 使用	3.3.3
Spark 监控	4
Web Interfaces	4.1
Spark Metrics	4.2
Spark 调优	5
Spark Core	6

Context	6.1
RDD	6.2
Key-Value Pairs RDD	6.2.1
Transform	6.3
Action	6.4
Persist & Cache	6.5
Spark Streaming	7
DStream	7.1

Spark运维实战

本书参考Spark官方文档和源码，通过本书你将精通Spark的安装、配置、监控和调优。

Apache Spark

Spark的来源

Spark ecological environment

Spark

Spark Install

Spark 编译

有三种方式：SBT、MAVEN、make-distribution.sh。SBT、MAVEN两种方式打出来的包比较大，不适合部署使用。因此我们通常使用第三种方式打包。

官方已经提供安装包了，为什么要自己编译？

Spark能同Hadoop进行交互，而Hadoop的厂商比较多有很多商业版。Spark官方提供的安装包不一定和我们的Hadoop集群版本相同，如果不相同就有可能出现莫名其妙的错误。这时，我们手工指定相应版本进行编译是最好选择。

SBT编译

```
sbt/sbt clean assembly
```

MAVEN编译

由于MAVEN工具默认的内存比较小，需要先调大其占用的内存上限：

```
export MAVEN_OPTS="-Xmx2g -XX:MaxPermSize=512M -XX:ReservedCodeCacheSize=512m"
```

打包

```
mvn clean assembly:assembly
```

make-distribution.sh构建安装包

该脚本会使用MAVEN进行编译，然后打成一个tgz包。

脚本的使用方法：

```
./make-distribution.sh --help
```

打包：


```
./make-distribution.sh --tgz --with-tachyon
```

Hadoop版本对应的MAVEN Profile

Hadoop version	Profile required
0.23.x	hadoop-0.23
1.x to 2.1.x	(none)
2.2.x	hadoop-2.2
2.3.x	hadoop-2.3
2.4.x	hadoop-2.4

Yarn版本对应的MAVEN Profile

YARN version	Profile required
0.23.x to 2.1.x	yarn-alpha
2.2.x and later	yarn

Hive对应的MAVEN Profile

在构造脚本后面添加 -Phive便可

自定义Hadoop版本

如果要构建hortonworks Hadoop 2.4.0.2.1.4.0-632，所对应的Hadoop版本是2.4.x。因此，相应的Profile为-Phadoop-2.4 -Pyarn。

编译方式：

SBT

```
sbt clean assembly -Phive -Phadoop-2.4 -Pyarn -Dhadoop.version=2.4.0.2.1.4.0-632
```

Maven


```
export MAVEN_OPTS="-Xmx2g -XX:MaxPermSize=512M -XX:ReservedCodeCacheSize=512m"
```

```
mvn clean assembly:assembly
```

make-distribution.sh

1.1.x 使用

```
./make-distribution.sh --tgz --with-tachyon -Phadoop-2.4 -Pyarn -Phive -Dhadoop.version=2
```



对于1.1.x以前的版本使用：

```
./make-distribution.sh --hadoop 2.4.0.2.1.4.0-632 --with-yarn --with-tachyon --tgz
```

如果yarn的版本和Hadoop的版本不一致可添加

```
-Dyarn.version=2.4.0.2.1.4.0-632
```

Spark Deploy Type

单机运行

用来做一些简单测试，学习使用相关工具。

Standalone

Spark本身提供的资源管理器，可以直接运行。

提供HA功能

Yarn

Spark和Hadoop YARN集成，运行Hadoop集群中。由YARN提供资源分配管理。

提供运行spark应用的jar包

Mesos

另一种资源管理系统

Amazon EC2 / ElasticMapReduce

Spark Single

Spark Standalone

Spark Standalone采用了Master/Slaves架构的集群模式，因此，存在着Master单点故障。

Spark提供了两种单点故障的解决方案：

- 基于文件系统的单点恢复
- 基于ZooKeeper的Standby Masters

基于文件系统的单点恢复

基于ZooKeeper的Standby Masters

无HA安装配置

此模式主要用来做开发，因为开发时应用运行频率高，而且对Master故障的影响不大，最主要的是出现故障重新运行便可，不需要恢复。

主机规划

HDP125作为Master节点，其余主机作为Worker节点。

关闭iptables

关闭正在运行的iptables防火墙：

```
service iptables stop
```

关闭开机自动启动iptables：

```
chkconfig iptables off
```

关闭SELinux

关闭自在运行的SELinux：

```
setenforce 0
```

修改配置文件，关闭开机自己启动SELinux：

```
vi /etc/selinux/config  
SELINUX=disabled
```

配置主机名和映射：

注意：主机名只能用英文字母、数字、“-”。不能使用下划线“_”，会出现问题。

映射

Spark通过主机名来进行互相访问，通过修改/etc/hosts文件可配置本地主机名映射关系，在hosts文件中添加计算机的名称和IP的对应关系，如在本机中添加master的主机（假设IP为172.16.219.125），在末尾添加内容为：172.16.219.125 HDP125

所有主机都在/etc/hosts添加：

```
172.16.219.125 HDP125
172.16.219.126 HDP126
172.16.219.127 HDP127
172.16.219.128 HDP128
```

修改主机名

修改本次运行期间的主机名：

```
hostname master
```

修改/etc/sysconfig/network文件，将主机名改为master：

```
HOSTNAME=master
```

操作系统启动的时候，会读取该文件并设置主机名。因此，修改后不会立即生效。只有当系统重启后，主机名便会生效。

在所有主机上执行上面两步，并把master替换成相应主机名。

用户：

在所有主机上添加用户：

```
groupadd spark
useradd spark -g spark
```

SSH无密码登录

Master到Worker节点需要配置SSH无密码登录。

在Master生成公私钥对：

```
su spark
ssh-keygen -t dsa -P '' -f ~/.ssh/id_dsa
```

在其它主机上创建~/.ssh文件夹：

```
su spark
mkdir ~/.ssh
chmod 700 ~/.ssh
```

把公钥~/.ssh/dsa.pub发送到其它主机：

```
scp ~/.ssh/dsa.pub spark@HDPXXX:~/.ssh/id_dsa.pub
```

测试是否成功：

```
ssh HDPxxx
```

安装Java的JDK

解压：

```
tar -zxvf jdk-7u51-linux-x64.tar.gz -C /opt
ln -s /opt/jdk1.7.0_51 /opt/jdk
```

配置环境变量：

```
#JDK setting
export JAVA_HOME=/opt/jdk
export CLASSPATH=.:$JAVA_HOME/jre/lib/rt.jar:$JAVA_HOME/lib/tools.jar
export PATH=$JAVA_HOME/bin:$PATH
```

安装配置Spark

解压与权限


```
tar -zxvf spark-1.1.0-bin-2.4.0.2.1.4.0-632.tgz -C /opt
ln -s /opt/spark-1.1.0-bin-2.4.0.2.1.4.0-632 /opt/spark
chown -R spark:spark /opt/spark*
```

配置worker

```
vi conf/slaves
```

```
HDP125
```

```
HDP126
```

```
HDP127
```

注：每行一个Worker

配置spark-env.sh

```
cp conf/spark-env.sh.template conf/spark-env.sh
```

```
vi conf/spark-env.sh
```

```
export JAVA_HOME=/opt/jdk
```

```
export SPARK_MASTER_IP=ES122
```

可以为Master和Worker的CPU核心和内存大小进行定制。

- SPARK_MASTER_IP：设置Master的IP
- SPARK_DAEMON_MEMORY：设置Master和Worker守护进程内存大小
- SPARK_WORKER_CORES：设置Spark应用在Worker中可以使用的CPU核数
- SPARK_WORKER_MEMORY：设置Spark应用在Worker中可以使用的内存总量
- SPARK_MASTER_WEBUI_PORT：设置Master的Web UI端口
- SPARK_WORKER_WEBUI_PORT：设置Worker的Web UI端口

更多配置参考：[Spark Standalone配置属性](#)

启动

启动集群

在master上

```
su spark
cd /opt/spark
sbin/start-all.sh
```

启动Worker

```
./sbin/start-slave.sh spark://IP:PORT
```

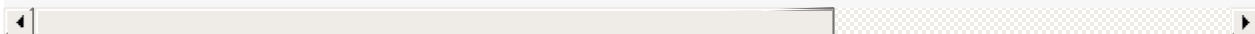
或

```
./bin/spark-class org.apache.spark.deploy.worker.Worker spark://IP:PORT
```

测试

提交Application :

```
bin/spark-submit --class org.hansight.spark.examples.SogouQTotal --master spark://HDP125:
```



基于文件系统的单点恢复

此模式下虽然可以恢复，但需要人工参与。因此，适合对高可用要求不高的场景，如果离线分析。

当Master挂掉后，手工启动Master仍然能继续执行原来的任务。当然，也可以继续提交任务。

主机规划

HDP125作为Master节点，其余主机作为Worker节点。

关闭iptables

关闭正在运行的iptables防火墙：

```
service iptables stop
```

关闭开机自动启动iptables：

```
chkconfig iptables off
```

关闭SELinux

关闭自在运行的SELinux：

```
setenforce 0
```

修改配置文件，关闭开机自己启动SELinux：

```
vi /etc/selinux/config
```

```
SELINUX=disabled
```

配置主机名和映射：

注意：主机名只能用英文字母、数字、“-”。不能使用下划线“_”，会出现问题。

映射

Spark通过主机名来进行互相访问，通过修改/etc/hosts文件可配置本地主机名映射关系，在hosts文件中添加计算机的名称和IP的对应关系，如在本机中添加master的主机（假设IP为172.16.219.125），在末尾添加内容为：172.16.219.125 HDP125

所有主机都在/etc/hosts添加：

```
172.16.219.125 HDP125
172.16.219.126 HDP126
172.16.219.127 HDP127
172.16.219.128 HDP128
```

修改主机名

修改/etc/sysconfig/network文件，将主机名改为master。

```
HOSTNAME=master
```

操作系统启动的时候，会读取该文件并设置主机名。因此，修改后不会立即生效。只有当系统重启后，主机名便会生效。

也需要运行如下命令设置主机名，无须重启：

```
hostname master
```

在所有主机上执行上面两步，并把master替换成相应主机名。

用户：

在所有主机上添加用户：

```
groupadd spark
useradd spark -g spark
```

SSH无密码登录

Master到Worker节点需要配置SSH无密码登录。

在Master生成公私钥对：

```
su spark
ssh-keygen -t dsa -P '' -f ~/.ssh/id_dsa
```

在其它主机上创建~/.ssh文件夹：

```
su spark
mkdir ~/.ssh
chmod 700 ~/.ssh
```

把公钥~/.ssh/dsa.pub发送到其它主机：

```
scp ~/.ssh/dsa.pub spark@HDPXXX:~/.ssh/id_dsa.pub
```

测试是否成功：

```
ssh HDPxxx
```

安装Java的JDK

解压：

```
tar -zxvf jdk-7u51-linux-x64.tar.gz -C /opt
ln -s /opt/jdk1.7.0_51 /opt/jdk
```

配置环境变量：

```
#JDK setting
export JAVA_HOME=/opt/jdk
export CLASSPATH=.:$JAVA_HOME/jre/lib/rt.jar:$JAVA_HOME/lib/tools.jar
export PATH=$JAVA_HOME/bin:$PATH
```

安装配置Spark

解压与权限

```
tar -zxvf spark-1.1.0-bin-2.4.0.2.1.4.0-632.tgz -C /opt
ln -s /opt/spark-1.1.0-bin-2.4.0.2.1.4.0-632 /opt/spark
chown -R spark:spark /opt/spark*
```

配置worker

```
vi conf/slaves
```

```
HDP125
HDP126
HDP127
```

注：每行一个Worker

配置spark-env.sh

```
cp conf/spark-env.sh.template conf/spark-env.sh
```

```
vi conf/spark-env.sh
```

```
export JAVA_HOME=/opt/jdk
export SPARK_MASTER_IP=ES122
export SPARK_DAEMON_JAVA_OPTS="-Dspark.deploy.recoveryMode=FILESYSTEM -Dspark.deploy.reco
```

可以为Master和Worker的CPU核心和内存大小进行定制。

- SPARK_MASTER_IP：设置Master的IP
- SPARK_DAEMON_MEMORY：设置Master和Worker守护进程内存大小
- SPARK_WORKER_CORES：设置Spark应用在Worker中可以使用的CPU核数
- SPARK_WORKER_MEMORY：设置Spark应用在Worker中可以使用的内存总量
- SPARK_MASTER_WEBUI_PORT：设置Master的Web UI端口
- SPARK_WORKER_WEBUI_PORT：设置Worker的Web UI端口

更多配置参考：[Spark Standalone配置属性](#)

SPARK_DAEMON_JAVA_OPTS配置项：

System property	Meaning
spark.deploy.recoveryMode	设成FILESYSTEM来开启单节点恢复模式，（默认值：NONE）
spark.deploy.recoveryDirectory	Spark存储恢复状态的目录，Master能够访问

配置SPARK_HOME环境变量

```
vi /etc/profile
```

```
#spark
export SPARK_HOME=/opt/spark
export PATH=$SPARK_HOME/bin:$PATH
source /etc/profile
```

启动

启动集群

在master上

```
su spark
cd /opt/spark
sbin/start-all.sh
```

不用**sbin**脚本的方式

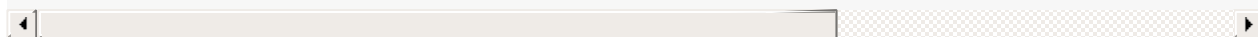
启动Worker

```
./bin/spark-class org.apache.spark.deploy.worker.Worker spark://IP:PORT
```

测试

提交**Application** :

```
bin/spark-submit --class org.hansight.spark.examples.SogouQTotal --master spark://HDP125:
```



Spark Standalone基于ZooKeeper的HA

适用于生产环境

Spark Standalone使用多个Master节点，通过ZooKeeper推举一个为Active（激活）状态，其它的均为Standby（备用）状态。

当Active Master节点挂掉后，ZooKeeper会从所有的Standby Mater中推举会一个新的Activer Master。新的Active Master恢复到旧有的Active Master的状态，然后恢复调度。从Activer Master失去作用，到新Active Master恢复调度可能需要1—2分钟。

故障恢复期间，只影响新应用程序的调用，已经运行的应用程序不会受到影响。

主机规划

- HDP125、HDP126作为Master节点
- HDP245、HDP246、HDP247作为ZooKeeper节点
- HDP127、HDP128作为Worker节点

注：Mater、ZooKeeper建议各自独占一台服务器，Worker节点可以动态添加。如果集群机器不多，也可以让Mater和ZooKeeper、Worker共用一个节点，但要配置好相关资源，如内存、CPU、网络。

关闭iptables

关闭正在运行的iptables防火墙：

```
service iptables stop
```

关闭开机自动启动iptables：

```
chkconfig iptables off
```

关闭SELinux

关闭自在运行的SELinux：


```
setenforce 0
```

修改配置文件，关闭开机自己启动SELinux：

```
vi /etc/selinux/config
```

内容

```
SELINUX=disabled
```

配置主机名和映射：

注意：主机名只能用英文字母、数字、“-”。不能使用下划线“_”，会出现问题。

映射

Spark通过主机名来进行互相访问，通过修改/etc/hosts文件可配置本地主机名映射关系，在hosts文件中添加计算机的名称和IP的对应关系，如在本机中添加master的主机（假设IP为172.16.219.125），在末尾添加内容为：172.16.219.125 HDP125

所有主机都在/etc/hosts添加：

```
172.16.219.125 HDP125
172.16.219.126 HDP126
172.16.219.127 HDP127
172.16.219.128 HDP128
172.16.219.245 HDP245
172.16.219.246 HDP246
172.16.219.247 HDP247
```

修改主机名

修改/etc/sysconfig/network文件，将主机名改为master。

```
HOSTNAME=master
```

操作系统启动的时候，会读取该文件并设置主机名。因此，修改后不会立即生效。只有当系统重启后，主机名便会生效。

也需要运行如下命令设置主机名，无须重启：

```
hostname master
```

在所有主机上执行上面两步，并把master替换成相应主机名。

用户：

在所有主机上添加用户：

```
groupadd spark
useradd spark -g spark

groupadd hadoop
useradd zookeeper -g zookeeper
```

SSH无密码登录

Master到Worker节点需要配置SSH无密码登录。

在Master生成公私钥对：

```
su spark
ssh-keygen -t dsa -P '' -f ~/.ssh/id_dsa
```

在其它主机上创建~/.ssh文件夹：

```
su spark
mkdir ~/.ssh
chmod 700 ~/.ssh
```

把公钥~/.ssh/dsa.pub发送到其它主机：

```
scp ~/.ssh/dsa.pub spark@HDPXXX:~/.ssh/id_dsa.pub
```

测试是否成功：

```
ssh HDPxxx
```

安装Java的JDK

解压：

```
tar -zxvf jdk-7u51-linux-x64.tar.gz -C /opt
ln -s /opt/jdk1.7.0_51 /opt/jdk
```

配置环境变量：

```
#JDK setting
export JAVA_HOME=/opt/jdk
export CLASSPATH=.:$JAVA_HOME/jre/lib/rt.jar:$JAVA_HOME/lib/tools.jar
export PATH=$JAVA_HOME/bin:$PATH
```

安装配置ZooKeeper

参考[3.1.1安装配置ZooKeeper](#)

安装配置Spark

解压与权限

```
tar -zxvf spark-1.1.0-bin-2.4.0.2.1.4.0-632.tgz -C /opt
ln -s /opt/spark-1.1.0-bin-2.4.0.2.1.4.0-632 /opt/spark
chown -R spark:spark /opt/spark*
```

配置worker

vi conf/slaves

```
HDP126
HDP127
```

注：每行一个Worker

配置spark-env.sh

```
cp conf/spark-env.sh.template conf/spark-env.sh
```

vi conf/spark-env.sh

```
export JAVA_HOME=/opt/jdk
export SPARK_DAEMON_JAVA_OPTS="-Dspark.deploy.recoveryMode=ZOOKEEPER -Dspark.deploy.zooke
```

可以为Master和Worker的CPU核心和内存大小进行定制。

- SPARK_MASTER_IP：设置Master的IP
- SPARK_DAEMON_MEMORY：设置Master和Worker守护进程内存大小
- SPARK_WORKER_CORES：设置Spark应用在Worker中可以使用的CPU核数
- SPARK_WORKER_MEMORY：设置Spark应用在Worker中可以使用的内存总量
- SPARK_MASTER_WEBUI_PORT：设置Master的Web UI端口
- SPARK_WORKER_WEBUI_PORT：设置Worker的Web UI端口

更多配置参考：[配置属性](#)

SPARK_DAEMON_JAVA_OPTS配置项：

System property	Meaning
spark.deploy.recoveryMode	设置成ZOOKEEPER，缺省值为 NONE
spark.deploy.zookeeper.url	ZooKeeper URL(如zk1:2181,zk2:2181...)
spark.deploy.zookeeper.dir	ZooKeeper存储恢复状态的目录，（默认为：/spark）

配置SPARK_HOME环境变量

vi /etc/profile

```
#spark
export SPARK_HOME=/opt/spark
export PATH=$SPARK_HOME/bin:$PATH
source /etc/profile
```

启动

启动集群

启动ZooKeeper：

参考[3.1.1安装配置ZooKeeper](#)

在一个master上启动：

```
su spark
cd /opt/spark
sbin/start-all.sh
```

在另一个master上启动：

```
su spark
cd /opt/spark
sbin/start-master.sh
```

不用**sbin**脚本的方式

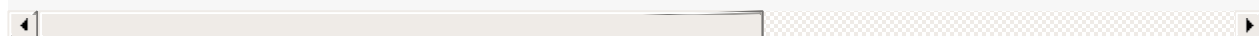
启动Worker

```
./bin/spark-class org.apache.spark.deploy.worker.Worker spark://IP:PORT
```

测试

提交**Application**：

```
bin/spark-submit --class org.hansight.spark.examples.SogouQTotal --master spark://HDP125:
```



Spark Standalone 配置

01. spark-env.sh支持的配置变量

Environment Variable	Meaning
SPARK_MASTER_IP	指定Master绑定的IP地址，例如公共IP
SPARK_MASTER_PORT	从另外一个端口启动master(默认: 7077)
SPARK_MASTER_WEBUI_PORT	Master的web UI端口 (默认: 8080)
SPARK_MASTER_OPTS	master守护进程的配置属性，格式“-Dx=y” (默认值: 无)。See below for a list of possible options
SPARK_LOCAL_DIRS	Directory to use for "scratch" space in Spark, including map output files and RDDs that get stored on disk. This should be on a fast, local disk in your system. It can also be a comma-separated list of multiple directories on different disks.
SPARK_WORKER_CORES	Total number of cores to allow Spark applications to use on the machine (default: all available cores).
SPARK_WORKER_MEMORY	Spark作业可使用的主机内存总量，默认格式1000M或者 2G (默认: 全部内在去年1GB);注意：每个作业自己的内存空间由属性spark.executor.memory决定。
SPARK_WORKER_PORT	启动Spark worker绑定的端口(默认：随机)
SPARK_WORKER_WEBUI_PORT	worker 的web UI 启动端口(默认: 8081)
SPARK_WORKER_INSTANCES	每台机器上运行worker数量 (默认: 1). 当你有一个非常强大的计算机，并且需要多个Spark worker进程的时候你可以修改这个默认值大于1. 如果你设置了这个值，要确保SPARK_WORKER_CORE明确限制每一个 worker的CPU核个数, 否则每个worker将尝试使用所有的CPU核
SPARK_WORKER_DIR	应用运行在该目录中，也包含日志 和临时空间(默认: SPARK_HOME/work);
SPARK_WORKER_OPTS	Configuration properties that apply only to the worker in the form "-Dx=y" (默认值: 无)。See below for a list of possible options.
SPARK_DAEMON_MEMORY	分配给Spark master和 worker 守护进程的内存空间 (默认: 512m)
SPARK_DAEMON_JAVA_OPTS	Spark master 和 worker守护进程的JVM 选项,格式“-Dx=y” (默认值: 无)
SPARK_PUBLIC_DNS	The public DNS name of the Spark master and workers (默认值: 无)

Note: The launch scripts do not currently support Windows. To run a Spark cluster on Windows, start the master and workers by hand.

02. SPARK_MASTER_OPTS支持下面的系统属性：

Property Name	Default	Meaning
spark.deploy.retainedApplications	200	The maximum number of completed applications to display. Older applications will be dropped from the UI to maintain this limit.
spark.deploy.retainedDrivers	200	The maximum number of completed drivers to display. Older drivers will be dropped from the UI to maintain this limit.
spark.deploy.spreadOut	true	Whether the standalone cluster manager should spread applications out across nodes or try to consolidate them onto as few nodes as possible. Spreading out is usually better for data locality in HDFS, but consolidating is more efficient for compute-intensive workloads.
spark.deploy.defaultCores	(infinite)	Default number of cores to give to applications in Spark's standalone mode if they don't set spark.cores.max. If not set, applications always get all available cores unless they configure spark.cores.max themselves. Set this lower on a shared cluster to prevent users from grabbing the whole cluster by default.
spark.worker.timeout	60	Number of seconds after which the standalone deploy master considers a worker lost if it receives no heartbeats.

03.SPARK_WORKER_OPTS支持下面的系统属性：

Property Name	Default	Meaning
spark.worker.cleanup.enabled	false	Enable periodic cleanup of worker / application directories. Note that this only affects standalone mode, as YARN works differently. Applications directories are cleaned up regardless of whether the application is still running.
spark.worker.cleanup.interval	1800 (30 minutes)	Controls the interval, in seconds, at which the worker cleans up old application work dirs on the local machine.
spark.worker.cleanup.appDataTtl	7 24 3600 (7 days)	The number of seconds to retain application work directories on each worker. This is a Time To Live and should depend on the amount of available disk space you have. Application logs and jars are downloaded to each application work dir. Over time, the work dirs can quickly fill up disk space, especially if you run jobs very frequently.

Spark on Yarn

Spark 0.6.0开始支持此功能

准备：

运行Spark-on-YARN需要Spark的二进制发布包。参考编译

配置：

环境变量：

SPARK_YARN_USER_ENV

用户可以在这个参数中设置Spark on YARN的环境变量，可以省略。

例如：SPARK_YARN_USER_ENV="JAVA_HOME=/jdk64,FOO=bar"。

SPARK_JAR

设置Spark jar在HDFS的位置。

例如：export SPARK_JAR=hdfs:///some/path.

在每台Hadoop NodeManager节点上设置变量

启动：

确保HADOOP_CONF_DIR或YARN_CONF_DIR所指向的目录包含Hadoop集群的配置文件。这些配置文件用来连接YARN的ResourceManager并写数据到DFS。此为提交任务的Spark安装，为了使用spark-submit工具。因此，只在此机器上配置便可。

有两种模式：

yarn-cluster：

Spark的driver运行YARN集群启动的一个application master进程中，client在初始化application后可以消失。

yarn-client :

Spark的driver运行在client进程中，而application master只用来向YARN申请资源。

不像Spark standalone和mesos模式，在那儿master地址使用指定的master参数；在YARN模式中，ResourceManager的地址从Hadoop配置文件中获取。因此，YARN模式中master参数简单的为“yarn-client”或“yarn-cluster”。

在yarn-cluster模式中启动一个application :

```
./bin/spark-submit --class path.to.your.Class --master yarn-cluster [options] <app jar> [
```

例如：

```
SPARK_JAR=hdfs://hansight/libs/spark-assembly-1.0.2-hadoop2.4.0.2.1.4.0-632.jar \  
./bin/spark-submit --class org.apache.spark.examples.SparkPI \  
  --master yarn-cluster \  
  --num-executors 3 \  
  --driver-memory 4g \  
  --executor-memory 2g \  
  --executor-cores 1 \  
  lib/spark-examples*.jar \  
  10
```

注：上面启动一个YARN客户端，该客户端启动默认的Application Master。SparkPI将作为一个子线程运行在Application Master中。client将定期的读取Application Master获取状态更新并把更新显示在控制台中。一旦你的application运行完成client会结束。

在yarn-client模式中启动一个application :

```
./bin/spark-submit --master yarn-client [options] <app jar> [app options]
```

只是把--master的参数值改为yarn-client，其它都与yarn-cluster相同。

添加其它JAR依赖

在yarn-cluster模式中，driver与client运行在不同的机器上。因此，SparkContext.addJar方法不会像client在本地模式那样开箱即用。为了使SparkContext.addJar可用，需要在启动命令参数--jars后面添加这些jar。例如：

```
$ ./bin/spark-submit --class my.main.Class \  
--master yarn-cluster \  
--jars my-other-jar.jar,my-other-other-jar.jar  
my-main-jar.jar  
app_arg1 app_arg2
```

Spark Shell

Spark-Shell是一个Spark Application，运行时需要向资源管理器申请资源，如Standalone Spark、YARN、Mesos。本例向Standalone Spark申请资源，所以在运行spark-shell时需要指向申请资源的Standalone Spark集群信息，其参数为MASTER。

启动spark-shell：

```
./bin/spark-shell --master spark://MASTER:PORT
```

如果未在spark-env.sh中申明MASTER，则使用命令MASTER=spark://bdp125:7077 bin/spark-shell启动；

如果已经在spark-env.sh中申明MASTER，则可以直接用bin/spark-shell启动。

集群模式：

```
MASTER=spark://`hostname`:7077 bin/spark-shell
```

或

```
bin/spark-shell --master spark://es122:7077
```

单机模式：

```
bin/spark-shell local[4]
```

加载一个text文件

Spark context available as sc.

连接到Spark的master之后，若集群中没有分布式文件系统，Spark会在集群中每一台机器上加载数据，所以要确保集群中的每个节点上都有完整数据。

[spam.data](#)

单机时：

```
var inFile = sc.textFile("./spam.data")
```

集群时：

```
import org.apache.spark.SparkFiles;
var file = sc.addFile("spam.data")
var inFile = sc.textFile(SparkFiles.get("spam.data"))
```

处理

文件的一行，按空格拆分，然后转成double。

```
var nums = inFile.map(x => x.split(" ").map(_.toDouble))
```

注：x => x.toDouble 等价于_.toDouble

查看：

```
inFile.first()
nums.first()
```

逻辑回归：

```
import org.apache.spark.util.Vector
case class DataPoint(x: Vector, y: Double)
def parsePoint(x: Array[Double]): DataPoint = {
  DataPoint(new Vector(x.slice(0, x.size-2 )), x(x.size -1 ))
}
```

Spark Submit

Usage: spark-submit [options] <app jar | python file> [app options]

Options:

<code>--master MASTER_URL</code>	spark://host:port, mesos://host:port, yarn, or local.
<code>--deploy-mode DEPLOY_MODE</code>	Whether to launch the driver program locally ("client") or on one of the worker machines inside the cluster ("cluster" (Default: client)).
<code>--class CLASS_NAME</code>	Your application's main class (for Java / Scala apps).
<code>--name NAME</code>	A name of your application.
<code>--jars JARS</code>	Comma-separated list of local jars to include on the driver and executor classpaths.
<code>--py-files PY_FILES</code>	Comma-separated list of .zip, .egg, or .py files to place on the PYTHONPATH for Python apps.
<code>--files FILES</code>	Comma-separated list of files to be placed in the working directory of each executor.
<code>--properties-file FILE</code>	Path to a file from which to load extra properties. If not specified, this will look for conf/spark-defaults.conf.
<code>--driver-memory MEM</code>	Memory for driver (e.g. 1000M, 2G) (Default: 512M).
<code>--driver-java-options</code>	Extra Java options to pass to the driver.bin
<code>--driver-library-path</code>	Extra library path entries to pass to the driver.
<code>--driver-class-path</code>	Extra class path entries to pass to the driver. Note that jars added with --jars are automatically included in the classpath.
<code>--executor-memory MEM</code>	Memory per executor (e.g. 1000M, 2G) (Default: 1G).
<code>--help, -h</code>	Show this help message and exit
<code>--verbose, -v</code>	Print additional debug output

Spark standalone with cluster deploy mode only:

<code>--driver-cores NUM</code>	Cores for driver (Default: 1).
<code>--supervise</code>	If given, restarts the driver on failure.

Spark standalone and Mesos only:

<code>--total-executor-cores NUM</code>	Total cores for all executors.
---	--------------------------------

YARN-only:

<code>--executor-cores NUM</code>	Number of cores per executor (Default: 1).
<code>--queue QUEUE_NAME</code>	The YARN queue to submit to (Default: "default").
<code>--num-executors NUM</code>	Number of executors to launch (Default: 2).
<code>--archives ARCHIVES</code>	Comma separated list of archives to be extracted into the working directory of each executor.

样例：

YARN :

```
./bin/spark-submit \  
--class org.apache.spark.examples.SparkPI \  
--master yarn-cluster \  
--num-executors 3 \  
--driver-memory 4g \  
--executor-memory 2g \  
--executor-cores 1 \  
lib/spark-examples*.jar \  
10
```

注解：

Spark相关软件

会根据实际情况，慢慢添加

暂时有以下软件：

- ZooKeeper
- Hadoop
- Kafka

zookeeper

ZooKeeper安装配置

下载：

```
wget http://archive.apache.org/dist/zookeeper/stable/zookeeper-3.4.6.tar.gz
```

解压与软链接：

```
tar -zxvf zookeeper-3.4.6.tar.gz -C /opt  
ln -s /opt/zookeeper-3.4.6 /opt/zookeeper  
chown -R zookeeper:hadoop /opt/zookeeper*
```

复制配置文件

```
cp /opt/zookeeper/zoo_sample.cfg /opt/zookeeper/zoo.cfg
```

修改配置

vi /opt/zookeeper/zoo.cfg

```
dataDir=/opt/zookeeper/data  
dataLogDir=/opt/zookeeper/logs  
clientPort=2181  
tickTime=2000  
initLimit=5  
syncLimit=2  
server.1=HDP245:2888:3888  
server.2=HDP246:2888:3888  
server.3=HDP247:2888:3888
```

在dataDir目录下创建myid文件，HDP245机器的内容为1，HDP246机器的内容为2，HDP247机器的内容为3，若有更多依此类推。

在HDP245的修改为：`mkdir -p /opt/zookeeper/data/ echo 1 > /opt/zookeeper/data/myid`

在HDP246、HDP247上把“echo 1”的“1”改成对应的值。

注：

dataDir：数据目录

dataLogDir：日志目录

clientPort：客户端连接端口

tickTime：Zookeeper 服务器之间或客户端与服务器之间维持心跳的时间间隔，也就是每个 tickTime 时间就会发送一个心跳。

initLimit：Zookeeper的Leader 接受客户端（Follower）初始化连接时最长能忍受多少个心跳时间间隔数。当已经超过 5个心跳的时间（也就是tickTime）长度后 Zookeeper 服务器还没有收到客户端的返回信息，那么表明这个客户端连接失败。总的时间长度就是 $5 * 2000 = 10$ 秒

syncLimit：表示 Leader 与 Follower 之间发送消息时请求和应答时间长度，最长不能超过多少个tickTime 的时间长度，总的时间长度就是 $2 * 2000 = 4$ 秒。

server.A=B：C：D：其中A是一个数字，表示这个是第几号服务器；B是这个服务器的ip地址；C表示的是这个服务器与集群中的Leader服务器交换信息的端口；D表示的是万一集群中的Leader服务器挂了，需要一个端口来重新进行选举，选出一个新的Leader，而这个端口就是用来执行选举时服务器相互通信的端口。如果是伪集群的配置方式，由于B都是一样，所以不同的Zookeeper实例通信端口号不能一样，所以要给它们分配不同的端口号。

启动与停止

启动：

```
/opt/zookeeper/bin/zkServer.sh start
```

停止：

```
/opt/zookeeper/bin/zkServer.sh stop
```

查看ZooKeeper树

Hadoop

hadoop 1.x 安装配置

在安装Hadoop的时候通常依赖一定的环境，如操作系统、JDK、SSH服务等。因此，在进行安装前，我们先进行这些环境的配置，然后安装Hadoop。Hadoop可以在单个节点上运行，使用进程模拟多台服务器。是完全分布式运行的一个特殊实例，元数据节点、数据节点、辅助元数据节点全部在一台服务器上。

主机规划

HDP125作为Master节点，其余主机作为Slave节点。

操作系统

Hadoop可以同时支持在Linux和Windows系统下进行安装，但是在windows系统下没有进行过大量的测试，不稳定容易出现問題。因此，我们使用Linux系统安装。

在国内企业中使用最多的Linux系统是CentOS，笔者就使用CentOS的最新版6.5（写作时）进行安装。

关闭iptables

关闭正在运行的iptables防火墙：

```
service iptables stop
```

关闭开机自动启动iptables：

```
chkconfig iptables off
```

关闭SELinux

关闭正在运行的SELinux：

```
setenforce 0
```

修改配置文件，关闭开机自己启动SELinux：

vi /etc/selinux/config

```
SELINUX=disabled
```

配置主机名和映射：

注意：主机名只能用英文字母、数字、“-”。不能使用下划线“_”，会出现问题。

映射

Spark通过主机名来进行互相访问，通过修改/etc/hosts文件可配置本地主机名映射关系，在hosts文件中添加计算机的名称和IP的对应关系，如在本机中添加master的主机（假设IP为172.16.219.125），在末尾添加内容为：172.16.219.125 HDP125

所有主机都在/etc/hosts添加：

```
172.16.219.125 HDP125
172.16.219.126 HDP126
172.16.219.127 HDP127
172.16.219.128 HDP128
```

修改主机名

修改本次运行期间的主机名：

```
hostname HDP125
```

修改/etc/sysconfig/network文件，将主机名改为HDP125：

```
HOSTNAME=HDP125
```

操作系统启动的时候，会读取该文件并设置主机名。因此，修改后不会立即生效。只有当系统重启后，主机名便会生效。

在所有主机上执行上面两步，并把HDP125替换成相应主机名。

用户：

在所有主机上添加用户：

```
groupadd hadoop
useradd hadoop -g hadoop
```

SSH无密码登录

Master到Slaves节点需要配置SSH无密码登录。

在Master生成公私钥对：

```
su hadoop
ssh-keygen -t dsa -P '' -f ~/.ssh/id_dsa
```

在其它主机上创建~/.ssh文件夹：

```
su hadoop
mkdir ~/.ssh
chmod 700 ~/.ssh
```

把公钥~/.ssh/dsa.pub发送到其它主机：

```
scp ~/.ssh/dsa.pub hadoop@HDPXXX:~/.ssh/id_dsa.pub
```

测试是否成功：

```
ssh HDPxxx
```

安装Java的JDK

解压：

```
tar -zxvf jdk-7u51-linux-x64.tar.gz -C /opt
ln -s /opt/jdk1.7.0_51 /opt/jdk
```

配置环境变量：


```
#JDK setting
export JAVA_HOME=/opt/jdk
export CLASSPATH=.:$JAVA_HOME/jre/lib/rt.jar:$JAVA_HOME/lib/tools.jar
export PATH=$JAVA_HOME/bin:$PATH
```

配置文件句柄数

修改本次运行期间的句柄数：

```
ulimit -n 65535
```

修改操作系统重启时默认的句柄数：

```
vi /etc/security/limits.conf
```

```
\*      hard    nofile   65535
\*      soft    nofile   65535
```

Hadoop的下载与解压

到hadoop.apache.org上下载Hadoop的安装文件，笔者使用的是“hadoop-1.2.1.tar.gz”。

在发布式安装模式下，所有服务器Hadoop的安装目录需要一样。笔者安装的位置为/opt/hadoop，使用解压命令如下：

```
tar -zxvf hadoop-1.2.1.tar.gz -C /opt/
mv /opt/hadoop-1.2.1 /opt/hadoop
chown -R hadoop:hadoop /opt/hadoop
su hadoop
```

配置hadoop-env.sh

修改hadoop目录下conf/hadoop-env.sh的环境变量，在末尾添加：

```
export JAVA_HOME=/opt/jdk
```

注：此处采用最小配置

配置core-site.xml

修改hadoop目录下conf/core-site.xml的配置文件，在标签中添加如下内容：

```
fs.default.name hdfs://HDP125:9000
hadoop.tmp.dir /home/${user.name}/tmp
fs.trash.interval 1440
```

配置hdfs-site.xml

修改hadoop目录下conf/hdfs-site.xml的配置文件，在标签中添加如下内容：

```
<property>
  <name>dfs.replication</name>
  <value>3</value>
</property>
<property>
  <name>dfs.name.dir</name>
  <value>/home/${user.name}/dfs_name</value>
</property>
<property>
  <name>dfs.data.dir</name>
  <value>/home/${user.name}/dfs_data</value>
</property>
<property>
  <name>dfs.support.append</name>
  <value>true</value>
</property>
<property>
  <name>dfs.datanode.max.xcievers</name>
  <value>4096</value>
</property>
```

说明：

dfs.replication：文件复本数

dfs.namenode.name.dir：设置NameNode存储元数据(fsimage)的本地文件系统位置

dfs.datanode.data.dir：设置DataNode存储数据的本地文件系统位置

dfs.support.append：设置HDFS是否支持在文件末尾追加数据

dfs.datanode.max.xcievers：设置datanode可以创建的最大xcievers数

配置mapred-site.xml

修改hadoop目录下conf/mapred-site.xml的配置文件，在标签中添加如下内容：

```
<property>
  <name>mapred.job.tracker</name>
  <value>HDP125:9001</value>
</property>
<property>
  <name>mapred.system.dir</name>
  <value>/home/${user.name}/mapred/system</value>
</property>
<property>
  <name>mapred.local.dir</name>
  <value>/home/${user.name}/mapred/local</value>
</property>
<property>
  <name>mapreduce.job.counters.limit</name>
  <value>1400</value>
</property>
<property>
  <name>mapred.map.tasks.speculative.execution</name>
  <value>false</value>
</property>
<property>
  <name>mapred.reduce.tasks.speculative.execution</name>
  <value>false</value>
</property>
```

注：属性名为mapred.job.tracker来配置job tracker交互端口，

配置辅助名称节点：

修改hadoop目录conf/masters的配置文件，修改内容为：

```
HDP125
```

配置子节点：

修改hadoop目录conf/slave文件，每行一个节点，伪分布式修改内容为：

```
HDP126
HDP127
HDP128
```

把配置好的Hadoop程序复制到其它节点

```
scp -r /opt/hadoop root@HDP126:/opt/  
chown -R hadoop:hadoop /opt/hadoop
```

启动

格式化

第一次在使用一个分布式文件系统，需要对其进行格式化：

```
./bin/hadoop namenode -format
```

启动与停止

启动Hadoop守护进程：

```
./bin/start-all.sh
```

Hadoop守护进程的日志写入到\${HADOOP_LOG_DIR}目录（默认为logs下）。

停止Hadoop：

完成全部操作后，停止Hadoop守护进程：

```
bin/stop-all.sh
```

使用

将输入文件拷贝到分布式文件系统：

```
bin/hadoop fs -put ./conf/core-site.xml input
```

运行发行版提供的**WordCount**示例程序：

```
./bin/hadoop jar hadoop-examples-1.2.1.jar wordcount input output
```

查看输出文件：

在分布式文件系统上查看输出文件：

```
$ bin/hadoop fs -cat output/*
```

或者将输出文件从分布式文件系统拷贝到本地文件系统查看：

```
bin/hadoop fs -get output output  
cat output/*
```

Hadoop自带的WEB管理界面

浏览NameNode和JobTracker的WEB管理界面，它们的地址默认为：

NameNode - <http://master:50070/dfshealth.jsp>

JobTracker - <http://master:50030/jobtracker.jsp>

hadoop 2.x 安装配置

在安装Hadoop的时候通常依赖一定的环境，如操作系统、JDK、SSH服务等。因此，在进行安装前，我们先进行这些环境的配置，然后安装Hadoop。Hadoop可以在单个节点上运行，使用进程模拟多台服务器。是完全分布式运行的一个特殊实例，元数据节点、数据节点、辅助元数据节点全部在一台服务器上。

主机规划

HDP125作为Master节点，其余主机作为Slave节点。

操作系统

Hadoop可以同时支持在Linux和Windows系统下进行安装，但是在windows系统下没有进行过大量的测试，不稳定容易出现问題。因此，我们使用Linux系统安装。

在国内企业中使用最多的Linux系统是CentOS，笔者就使用CentOS的最新版6.5（写作时）进行安装。

关闭iptables

关闭正在运行的iptables防火墙：

```
service iptables stop
```

关闭开机自动启动iptables：

```
chkconfig iptables off
```

关闭SELinux

关闭正在运行的SELinux：

```
setenforce 0
```

修改配置文件，关闭开机自己启动SELinux：

```
vi /etc/selinux/config
```

```
SELINUX=disabled
```

配置主机名和映射：

注意：主机名只能用英文字母、数字、“-”。不能使用下划线“_”，会出现问题。

映射

Spark通过主机名来进行互相访问，通过修改/etc/hosts文件可配置本地主机名映射关系，在hosts文件中添加计算机的名称和IP的对应关系，如在本机中添加master的主机（假设IP为172.16.219.125），在末尾添加内容为：172.16.219.125 HDP125

所有主机都在/etc/hosts添加：

```
172.16.219.125 HDP125
172.16.219.126 HDP126
172.16.219.127 HDP127
172.16.219.128 HDP128
```

修改主机名

修改本次运行期间的主机名：

```
hostname HDP125
```

修改/etc/sysconfig/network文件，将主机名改为HDP125：

```
HOSTNAME=HDP125
```

操作系统启动的时候，会读取该文件并设置主机名。因此，修改后不会立即生效。只有当系统重启后，主机名便会生效。

在所有主机上执行上面两步，并把HDP125替换成相应主机名。

用户：

在所有主机上添加用户：

```
groupadd hadoop
useradd hadoop -g hadoop
```

SSH无密码登录

Master到Slaves节点需要配置SSH无密码登录。

在Master生成公私钥对：

```
su hadoop
ssh-keygen -t dsa -P '' -f ~/.ssh/id_dsa
```

在其它主机上创建~/.ssh文件夹：

```
su hadoop
mkdir ~/.ssh
chmod 700 ~/.ssh
```

把公钥~/.ssh/dsa.pub发送到其它主机：

```
scp ~/.ssh/dsa.pub hadoop@HDPXXX:~/.ssh/id_dsa.pub
```

测试是否成功：

```
ssh HDPxxx
```

安装Java的JDK

解压：

```
tar -zxvf jdk-7u51-linux-x64.tar.gz -C /opt
ln -s /opt/jdk1.7.0_51 /opt/jdk
```

配置环境变量：


```
#JDK setting
export JAVA_HOME=/opt/jdk
export CLASSPATH=.:$JAVA_HOME/jre/lib/rt.jar:$JAVA_HOME/lib/tools.jar
export PATH=$JAVA_HOME/bin:$PATH
```

配置文件句柄数

修改本次运行期间的句柄数：

```
ulimit -n 65535
```

修改操作系统重启时默认的句柄数：

```
vi /etc/security/limits.conf
```

```
\*      hard    nofile   65535
\*      soft    nofile   65535
```

Hadoop的下载与解压

到hadoop.apache.org上下载Hadoop的安装文件，笔者使用的是“hadoop-2.5.1.tar.gz”。

在发布式安装模式下，所有服务器Hadoop的安装目录需要一样。笔者安装的位置为/opt/hadoop，使用解压命令如下：

```
tar -zxvf hadoop-2.5.1.tar.gz -C /opt/
mv /opt/hadoop-2.5.1 /opt/hadoop
chown -R hadoop:hadoop /opt/hadoop
su hadoop
```

配置hadoop-env.sh

修改hadoop目录下conf/hadoop-env.sh的环境变量，在末尾添加：

```
export JAVA_HOME=/opt/jdk
```

注：此处采用最小配置

配置core-site.xml

修改hadoop目录下conf/core-site.xml的配置文件，在标签中添加如下内容：

```
<property>
  <name>fs.default.name</name>
  <value>hdfs://HDP125:9000</value>
</property>
<property>
  <name>hadoop.tmp.dir</name>
  <value>/home/${user.name}/tmp</value>
</property>
<property>
  <name>fs.trash.interval</name>
  <value>1440</value>
</property>
```

说明：

fs.defaultDFS：设置NameNode的IP和端口

hadoop.tmp.dir：设置Hadoop临时目录，（默认/tmp，机器重启会丢失数据！）

fs.trash.interval：开启Hadoop回收站

配置hdfs-site.xml

修改hadoop目录下conf/hdfs-site.xml的配置文件，在标签中添加如下内容：

```
<property>
  <name>dfs.replication</name>
  <value>1</value>
</property>
<property>
  <name>dfs.namenode.name.dir</name>
  <value>/home/${user.name}/dfs_name</value>
</property>
<property>
  <name>dfs.datanode.data.dir</name>
  <value>/home/${user.name}/dfs_data</value>
</property>
<property>
  <name>dfs.support.append</name>
  <value>true</value>
</property>
<property>
  <name>dfs.datanode.max.xcievers</name>
  <value>4096</value>
</property>
```

说明：

dfs.replication：文件副本数

dfs.namenode.name.dir：设置NameNode存储元数据(fsimage)的本地文件系统位置

dfs.datanode.data.dir：设置DataNode存储数据的本地文件系统位置

dfs.support.append：设置HDFS是否支持在文件末尾追加数据

dfs.datanode.max.xcievers：设置datanode可以创建的最大xcievers数

配置yarn-site.xml

修改hadoop目录下etc/hadoop/yarn-site.xml的配置文件，在标签中添加如下内容：

```
<property>
  <name>yarn.nodemanager.aux-services</name>
  <value>mapreduce_shuffle</value>
</property>
<property>
  <name>yarn.resourcemanager.hostname</name>
  <value>master</value>
</property>
<property>
  <name>yarn.nodemanager.aux-services.mapreduce.shuffle.class</name>
  <value>org.apache.hadoop.mapred.ShuffleHandler</value>
</property>
<property>
  <name>yarn.timeline-service.hostname</name>
  <value>master</value>
</property>
```

注：属性名为mapred.job.tracker来配置job tracker交互端口

配置mapred-site.xml

修改hadoop目录下etc/hadoop/mapred-site.xml的配置文件，在标签中添加如下内容：

```
<property>
  <name>mapreduce.framework.name</name>
  <value>yarn</value>
</property>
```

配置子节点：

修改hadoop目录etc/hadoop/slave文件，每行一个节点，伪分布式修改内容为：

```
HDP126
HDP127
HDP128
```

把配置好的Hadoop程序复制到其它节点

```
scp -r /opt/hadoop root@HDP126:/opt/
chown -R hadoop:hadoop /opt/hadoop
```

启动

格式化

第一次使用一个分布式文件系统，需要对其进行格式化：

```
./bin/hadoop namenode -format
```

启动Hadoop守护进程：

```
./sbin/start-all.sh
```

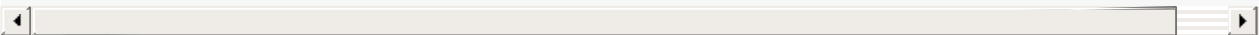
Hadoop守护进程的日志写入到\${HADOOP_LOG_DIR}目录（默认为logs下）。

停止Hadoop守护进程：

```
sbin/stop-all.sh
```

验证集群

```
bin/hadoop jar share/hadoop/mapreduce/hadoop-mapreduce-examples-2.5.1.jar randomwriter ou
```



Hadoop自带的WEB管理界面

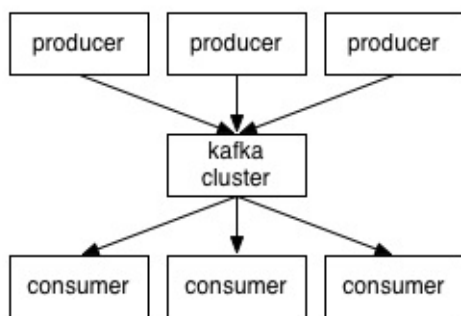
浏览NameNode和JobTracker的WEB管理界面，它们的地址默认为：

NameNode - <http://HDP125:50070/dfshealth.jsp>

Yarn - <http://HDP125:8088>

Kafka

kafka 官网地址：<http://kafka.apache.org>，是一款分布式消息发布和订阅的系统，具有高性能和高吞吐率。



消息的发布（publish）称作producer，消息的订阅（subscribe）称作consumer，中间的存储阵列称作broker。

多个broker协同合作，producer、consumer和broker三者之间通过zookeeper来协调请求和转发。

producer产生和推送(push)数据到broker，consumer从broker拉取(pull)数据并进行处理。

broker端不维护数据的消费状态，提升了性能。

直接使用磁盘进行存储，线性读写，速度快：避免了数据在JVM内存和系统内存之间的复制，减少耗性能的创建对象和垃圾回收。

Kafka使用scala编写，可以运行在JVM上。

kafka install

主机规划

HDP125、HDP126、HDP127、HDP128都作为KAFKA的Broker节点。

操作系统

Hadoop可以同时支持在Linux和Windows系统下进行安装，但是在windows系统下没有进行过大量的测试，不稳定容易出现問題。因此，我们使用Linux系统进行安装。

在国内企业中使用最多的Linux系统是CentOS，笔者就使用CentOS的最新版6.5（写作时）进行安装。

关闭iptables

关闭正在运行的iptables防火墙：

```
service iptables stop
```

关闭开机自动启动iptables：

```
chkconfig iptables off
```

关闭SELinux

关闭自在运行的SELinux：

```
setenforce 0
```

修改配置文件，关闭开机自己启动SELinux：

```
vi /etc/selinux/config
```

```
SELINUX=disabled
```

配置主机名和映射：

注意：主机名只能用英文字母、数字、“-”。不能使用下划线“_”，会出现问题。

映射

Spark通过主机名来进行互相访问，通过修改/etc/hosts文件可配置本地主机名映射关系，在hosts文件中添加计算机的名称和IP的对应关系，如在本机中添加master的主机（假设IP为172.16.219.125），在末尾添加内容为：172.16.219.125 HDP125

所有主机都在/etc/hosts添加：

```
172.16.219.125 HDP125
172.16.219.126 HDP126
172.16.219.127 HDP127
172.16.219.128 HDP128
```

修改主机名

修改本次运行期间的主机名：

```
hostname HDP125
```

修改/etc/sysconfig/network文件，将主机名改为HDP125：

```
HOSTNAME=HDP125
```

操作系统启动的时候，会读取该文件并设置主机名。因此，修改后不会立即生效。只有当系统重启后，主机名便会生效。

在所有主机上执行上面两步，并把HDP125替换成相应主机名。

用户：

在所有主机上添加用户：

```
groupadd kafka
useradd kafka -g kafka
```


ZooKeeper集群的安装

参考[3.1 ZooKeeper安装配置](#)

Kafka集群

解压与软链接

```
tar -zxvf kafka_2.10-0.8.1.1.tgz -C /opt
ln -s /opt/kafka_2.10-0.8.1.1 /opt/kafka
chown -R kafka:kafka /opt/kafka /opt/kafka_2.10-0.8.1.1
```

配置

vi /opt/kafka/config/server.properties

```
broker.id=1
port=9092
log.dirs=/opt/kafka/tmp
host.name=HDP125
zookeeper.connect=HDP125:2181,HDP126:2181,HDP127:2181
```

注：把host.name改成相应的主机名；broker.id不能相同，逐个加1便可。

更多配置项参考[kafka配置属性](kafka_config_properties.md)

启动

启动ZooKeeper

参考[3.1 ZooKeeper安装配置](#)

启动Kafka

```
/bin/kafka-server-start.sh -daemon ../config/server.properties
```

测试

使用Kafka自带的脚本，启动基于Console的producer和consumer。

启动**console consumer**

启动**console producer**

kafka config properties

属性名	默认值	说明
broker.id	0	多个Kafka服务不能相同
port	9092	KAFKA绑定的端口
zookeeper.connect	localhost:2181	ZooKeeper的连接URL
zookeeper.connection.timeout.ms	1000000	ZooKeeper的超时连接时间
log.dirs	/tmp/kafka-logs	
#host.name=HDP125	无	主机名
#advertised.host.name	HDP125	
#advertised.port	9092	
num.network.threads	2	
num.io.threads	8	
socket.send.buffer.bytes	1048576	
socket.receive.buffer.bytes	1048576	
socket.request.max.bytes	104857600	
num.partitions	2	
#log.flush.interval.messages	10000	
#log.flush.interval.ms	1000	
log.retention.hours	168	
#log.retention.bytes	1073741824	
log.segment.bytes	536870912	
log.retention.check.interval.ms	60000	
log.cleaner.enable	false	

kafka usage

Spark Monitor（监控）

有几种方法监听Spark应用: web UIs, metrics, and external instrumentation.

Web Interfaces

每个SparkContext启动一个web UI，默认端口是4040，它显示关于应用的有用的信息。包括：

- 高度器stage和task和列表
- RDD大小和内存占用的概况
- 环境信息
- 正运行的executors信息

你可参通过web浏览器中打http://:4040来访问这个UI。如果有多个SparkContext运行在同一主机上，它们将绑定以4040开始连续的端口（如4041、4042等）

注意此信息默认只在应用运行时有效。如果想在应用运行完成后查看web UI，应该在启动应用前将spark.eventLog.enabled设置为true。Spark编码在UI中显示的Spark信息为Spark事件，并记录到持久化存储中。

Viewing After the Fact

Spark的Standalone模式集群管理器也有它自己的web UI。如果一个应用已经记录它生命周期过程中的事件，那么当它完成后通过Standalone master的Web UI会自动的展示。

如果Spark运行在Mesos或YARN，它仍然可能再次展现已经完成应用的UI，通过历史服务器（history server）提供已经存在的应用事件日志。你可以通过下面的命令启动history server：

```
./sbin/start-history-server.sh
```

When using the file-system provider class (see spark.history.provider below), the base logging directory must be supplied in the spark.history.fs.logDirectory configuration option, and should contain sub-directories that each represents an application's event logs. This creates a web interface at http://:18080 by default.

历史服务器可以做如下配置:

Environment Variable	Meaning
SPARK_DAEMON_MEMORY	server分配的内存 (默认: 512m).
SPARK_DAEMON_JAVA_OPTS	history server的JVM选项 (默认: 无).
SPARK_PUBLIC_DNS	history server公共地址. 如果没有设置, 链接到应用历史将使用服务器的内部地址, 成为损坏的链接 (默认: 无).
SPARK_HISTORY_OPTS	为history server配置spark.history.*选项 (默认: 无).

Property Name	Default	Meaning
spark.history.provider	org.apache.spark.deploy.history.FsHistoryProvider	应用历史后端实现的类名。目前只有一个实现, 由Spark提供, 它查看存储在文件系统里面的应用日志
spark.history.fs.updateInterval	10	history服务器显示信息的更新周期, 单位秒。每次更新检查日志的变化并把事件日志记录到持久化存储。
spark.history.retainedApplications	50	应用UI显示保留的数量。超过这个数量, 最旧的应用日志将被删除
spark.history.ui.port	18080	历史服务器的Web接口绑定的端口
spark.history.kerberos.enabled	false	表示历史服务器是否使用Kerberos登录。如果历史服务器访问的是安全的Hadoop集群上的HDFS文件, 这会非常有用。如果值为true, 它会使用spark.history.kerberos.principal和spark.history.kerberos.keytab的配置。
spark.history.kerberos.principal	(none)	历史服务器在Kerberos中的安全主体 (principal) 名称
spark.history.kerberos.keytab	(none)	Kerberos keytab文件的历史记录服务器的位置
spark.history.ui.acls.enable	false	授权用户查看应用程序信息的时候是否检查acl。如果启用, 无论应用程序的spark.ui.acls.enable怎么设置, 都要进行授权检查, 只有应用程序所有者和spark.ui.view.acls指定的用户可以查看应用程序信息;如果禁用, 不做任何检查。

注意所有这些UI中，可以点击表头进行排序，因此很容易识别慢task、数据倾斜(data skew)等。

Spark Metrics

Spark拥有一个基于Coda Hale Metrics Library的可配置Metrics系统，

这个Metrics系统通过配置文件进行配置。

Spark的Metrics系统允许用户把Spark metrics信息报告到各种各样的sink包含HTTP和JMX、CSV文件。

Spark的metrics系统解耦到每个Spark组件的实例中。每个实例里，你可以配置一组sink（metrics被报告到的地方）。

注：Coda Hale Metrics使用的Yammer.com开发的Metrics框架，官方网址是<https://github.com/dropwizard/metrics>，相关文档可在<https://dropwizard.github.io/metrics>查看。

配置文件

默认的配置文件的为“\$SPARK_HOME/conf/metrics.properties”，Spark启动时候会自动加载它。

如果想修改配置文件位置，可以使用java的运行时属性-Dspark.metrics.conf=xxx进行修改。。

Spark的Metrics系统支持的实例：

- master：Spark standalone模式的 master进程。
- applications：master进程里的一个组件，为各种应用作汇报。
- worker：Spark standalone模式的一个worker进程。
- executor：一个Spark executor。
- driver：Spark driver进程(该进程指创建SparkContext的那个)。

Spark的Metrics系统支持的Sink：

Sink指定metrics信息发送到哪，每个instance可以设置一个或多个Sink。

Sink源码位于包org.apache.spark.metrics.sink中。

ConsoleSink

记录Metrics信息到Console中。

名称	默认值	描述
class	org.apache.spark.metrics.sink.ConsoleSink	Sink类
period	10	轮询间隔
unit	seconds	轮询间隔的单位

CSVSink

定期的把Metrics信息导出到CSV文件中。

名称	默认值	描述
class	org.apache.spark.metrics.sink.CsvSink	Sink类
period	10	轮询间隔
unit	seconds	轮询间隔的单位
directory	/tmp	CSV文件存储的位置

JmxSink

可以通过JMX方式访问Metrics信息

名称	默认值	描述
class	org.apache.spark.metrics.sink.JmxSink	Sink类

MetricsServlet

名称	默认值	描述
class	org.apache.spark.metrics.sink.MetricsServlet	Sink类
path	VARIES*	Path prefix from the web server root
sample	false	Whether to show entire set of samples for histograms ('false' or 'true')

除master之外所有实例的默认路径为“/metrics/json”。

master有两个路径: “/metrics/aplications/json” App的信息、 “/metrics/master/json” Master的信息

GraphiteSink

名称	默认值	描述
class	org.apache.spark.metrics.sink.GraphiteSink	Sink 类
host	NONE	Graphite服务器主机名
port	NONE	Graphite服务器端口
period	10	轮询间隔
unit	seconds	轮询间隔的单位
prefix	EMPTY STRING	Prefix to prepend to metric name

GangliaSink

由于Licene限制，默认没有放到默认的build里面。需要自己打包

名称	默认值	描述
class	org.apache.spark.metrics.sink.GangliaSink	Sink 类
host	NONE	Ganglia 服务器的主机名或 multicast group
port	NONE	Ganglia服务器的端口
period	10	轮询间隔
unit	seconds	轮询间隔的单位
ttl	1	TTL of messages sent by Ganglia
mode	multicast	Ganglia网络模式('unicast' or 'multicast')

source :

第一种为Spark内部source, MasterSource、WorkerSource等, 它们会接收Spark组件的内部状态 ;

第二种为通用source, 如 : JvmSource, 它收集低级别的状态

配置方法

Spark Metrics中通过Source来收集状态信息，通常sink来控制状态信息的发送。

语法：

```
syntax: [instance].sink|source.[name].[options]=[value]
```

说明：

[instance]：

可取值为master、worker、executor、driver、applications、*等。

* 配置所有实例的属性，但能被其实例重置。

sink|source：

指定配置的是source，还是sink的属性

[name]：

指定source或sink的名称

[options]：

[value]

注意：

添加一个新sink，选项class的值为完整类名称（带包名）

一些Sink牵涉轮询时间，最短的轮询时间是1秒

精确的instance会覆盖*配置，如：master.sink.console.period会覆盖

*.sink.console.period

在 master 和 worker、client driver 中默认开启 MetricsServlet，所以你可以发送HTTP请求“/metrics/json”来获取JSON格式的所有metrics信息快照，请求“/metrics/master/json”和“/metrics/applications/json”来获取master实例和应用的信息。

示例

通过类名为所有实例开启 **ConsoleSink**

```
*.sink.console.class=org.apache.spark.metrics.sink.ConsoleSink
```

ConsoleSink的轮询周期 `*.sink.console.period=10`

```
*.sink.console.unit=seconds
```

Master实例重置轮询周期

```
master.sink.console.period=15
```

```
master.sink.console.unit=seconds
```

通过类名为所有实例开启 **JmxSink**

```
*.sink.jmx.class=org.apache.spark.metrics.sink.JmxSink
```

为所有实例开启 **CsvSink**

```
*.sink.csv.class=org.apache.spark.metrics.sink.CsvSink
```

CsvSink的轮询周期

```
*.sink.csv.period=1
```

```
*.sink.csv.unit=minutes
```

Polling directory for CsvSink `*.sink.csv.directory=/tmp/`

Worker实例重置轮询周期

```
worker.sink.csv.period=10
```

```
worker.sink.csv.unit=minutes
```

为 **master**和**worker**、**driver**、**executor**开启 **jvm source**

```
master.source.jvm.class=org.apache.spark.metrics.source.JvmSource
```

```
worker.source.jvm.class=org.apache.spark.metrics.source.JvmSource
```

```
driver.source.jvm.class=org.apache.spark.metrics.source.JvmSource
```

```
executor.source.jvm.class=org.apache.spark.metrics.source.JvmSource
```

参考文档：

```
${SPARK_HOME}/conf/metrics.properties.template  
http://spark.apache.org/docs/latest/monitoring.html
```

Spark optimize

1、垃圾回收

在conf/spark-env.sh中添加 `SPARK_JAVA_OPTS=-verberos:gc -XX:+PrintGCDetails -XX:+PrintGCTimeStamps` 如果发现集群耗费过多时间在垃圾回收上，可以通过 `spark.storage.memoryFaction` 调低RDD缓存的使用，这个值的默认值是0.66。

如果要运行的是耗时很久的Spark作业，可以通过设定 `spark.cleaner.ttl` 为一个非零值 `n`，表示每隔 `n` 秒清理一次元数据。默认Spark不会清理任何元数据。

2、persist()缓存

默认都以非序列化模式存储，以节省读取数据时的反序列化开销。如：`MEMORY_ONLY`
`MEMORY_AND_DISK` `DISK_ONLY`

如果存储级别后面添加了 `_SER` 后缀，Spark会在存储时对数据进行序列化，以节省存储空间。`MEMORY_ONLY_SER` `MEMORY_AND_DISK_SER` `DISK_ONLY_SER`

3、序列化

默认使用Java内置的序列化算法，建议使用KryoSerialier算法，针对性的做了优化。可以通过 `spark.serializer` 改成 `org.apache.spark.KryoSerializer` 来切换 // TODO 实验

Spark Core

Context

RDD

Key-Value Pairs RDD

0:0	1:0
map(func)	
filter(func)	
flatMap(func)	
mapPartitions(func)	
mapPartitionsWithIndex(func)	
filter(func)	
filter(func)	
filter(func)	
filter(func)	
filter(func)	
filter(func)	

Action

Persist & Cache

Spark Streaming

Spark Streaming是一个实时流处理框架，它把实时数据