

Spark 安装以及使用指导

任恒 2016212063

2019 年 5 月

Spark 安装以及使用指导

1. 关于 spark 基本概念

Google 在 2003 年和 2004 年先后发表了 Google 文件系统 GFS 和 MapReduce 编程模型两篇文章,. 基于这两篇开源文档,06 年 Nutch 项目子项目之一的 Hadoop 实现了两个强有力的开源产品:HDFS 和 MapReduce. Hadoop 成为了典型的大数据批量处理架构,由 HDFS 负责静态数据的存储,并通过 MapReduce 将计算逻辑分配到各数据节点进行数据计算和价值发现.之后以 HDFS 和 MapReduce 为基础建立了很多项目,形成了 Hadoop 生态圈.

而 Spark 则是 UC Berkeley AMP lab (加州大学伯克利分校 AMP 实验室)所开源的类 Hadoop MapReduce 的通用并行框架,专门用于大数据量下的迭代式计算.是为了跟 Hadoop 配合而开发出来的,不是为了取代 Hadoop, Spark 运算比 Hadoop 的 MapReduce 框架快的原因是因为 Hadoop 在一次 MapReduce 运算之后,会将数据的运算结果从内存写入到磁盘中,第二次 Mapredue 运算时在从磁盘中读取数据,所以其瓶颈在 2 次运算间的多余 IO 消耗. Spark 则是将数据一直缓存在内存中,直到计算得到最后的结果,再将结果写入到磁盘,所以多次运算的情况下, Spark 是比较快的. 其优化了迭代式工作负载。

关于 spark 的一些概念如下:

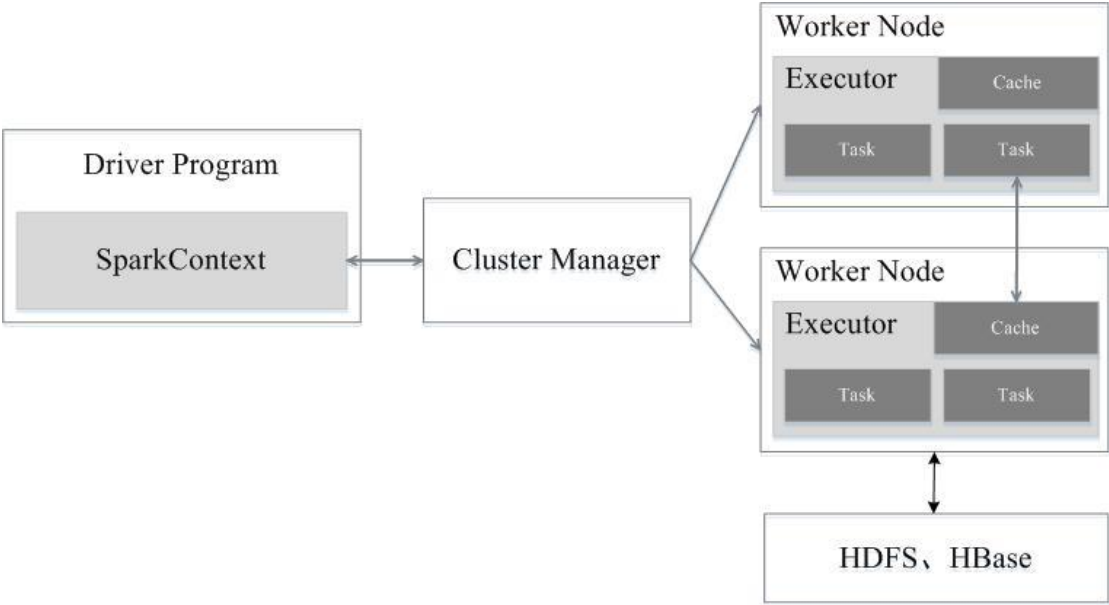
- **RDD:** 是弹性分布式数据集 (Resilient Distributed Dataset) 的简称,是分布式内存的一个抽象概念,提供了一种高度受限的共享内存模型;
- **DAG:** 是 Directed Acyclic Graph (有向无环图) 的简称,反映 RDD 之间的依赖关系;
- **Executor:** 是运行在工作节点 (Worker Node) 上的一个进程,负责运行任务,并为应用程序存储数据;
- **应用:** 用户编写的 Spark 应用程序;
- **任务:** 运行在 Executor 上的工作单元;
- **作业:** 一个作业包含多个 RDD 及作用于相应 RDD 上的各种操作;
- **阶段:** 是作业的基本调度单位,一个作业会分为多组任务,每组任务被称为“阶段”,或者也被称为“任务集”。

架构设计

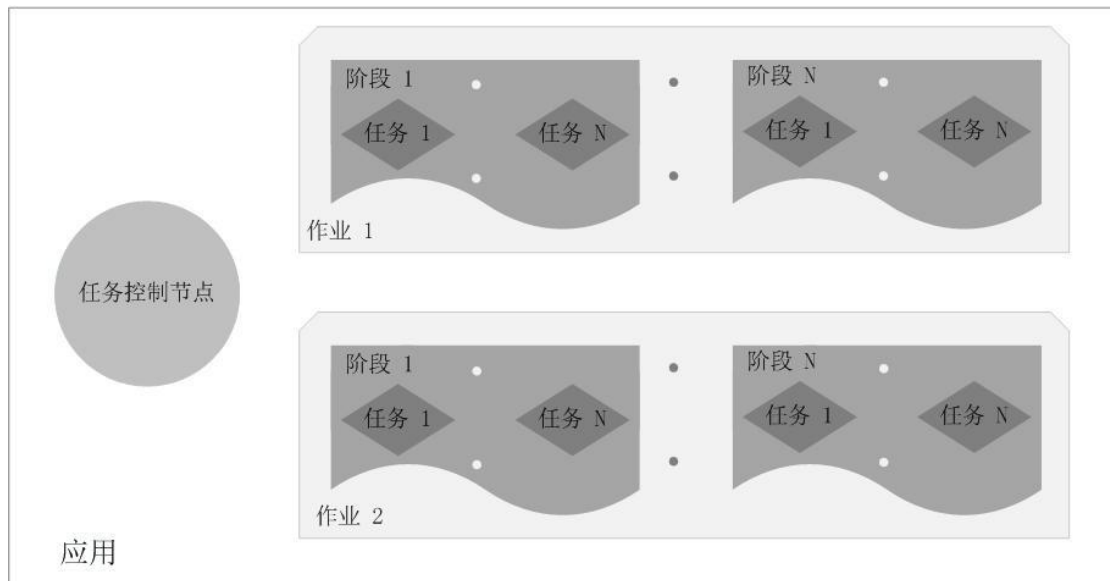
如图一所示，Spark 运行架构包括集群资源管理器（Cluster Manager）、运行作业任务的工作节点（Worker Node）、每个应用的任务控制节点（Driver）和每个工作节点上负责具体任务的执行进程（Executor）。其中，集群资源管理器可以是 Spark 自带的资源管理器，也可以是 YARN 或 Mesos 等资源管理框架。

与 Hadoop MapReduce 计算框架相比，Spark 所采用的 Executor 有两个优点：一是利用多线程来执行具体的任务（Hadoop MapReduce 采用的是进程模型），减少任务的启动开销；二是 Executor 中有一个 BlockManager 存储模块，会将内存和磁盘共同作为存储设备，当需要多轮迭代计算时，可以将中间结果存储到这个存储模块里，下次需要时，就可以直接读该存储模块里的数据，而不需要读写到 HDFS 等文件系统里，因而有效减少了 IO 开销；或者在交互式查询场景下，预先将表缓存到该存储系统上，从而提高读写 IO 性能。

总体而言，如图二所示，在 Spark 中，一个应用（Application）由一个任务控制节点（Driver）和若干个作业（Job）构成，一个作业由多个阶段（Stage）构成，一个阶段由多个任务（Task）组成。当执行一个应用时，任务控制节点会向集群管理器（Cluster Manager）申请资源，启动 Executor，并向 Executor 发送应用程序代码和文件，然后在 Executor 上执行任务，运行结束后，执行结果会返回给任务控制节点，或者写到 HDFS 或者其他数据库中。



图一



图二

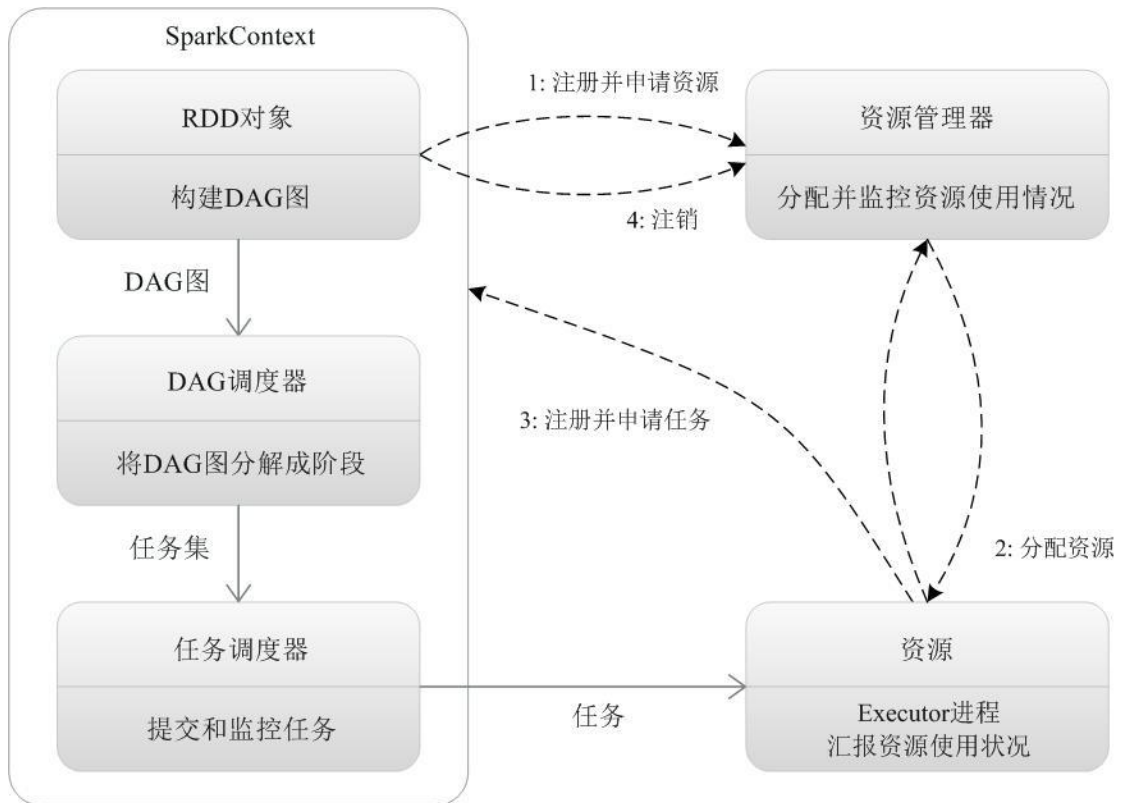
Spark 基本运行过程如图三：

(1) 当一个 Spark 应用被提交时，首先需要为这个应用构建起基本的运行环境，即由任务控制节点（Driver）创建一个 SparkContext，由 SparkContext 负责和资源管理器（Cluster Manager）的通信以及进行资源的申请、任务的分配和监控等。SparkContext 会向资源管理器注册并申请运行 Executor 的资源；

(2) 资源管理器为 Executor 分配资源，并启动 Executor 进程，Executor 运行情况将随着“心跳”发送到资源管理器上；

(3) SparkContext 根据 RDD 的依赖关系构建 DAG 图，DAG 图提交给 DAG 调度器（DAGScheduler）进行解析，将 DAG 图分解成多个“阶段”（每个阶段都是一个任务集），并且计算出各个阶段之间的依赖关系，然后把一个个“任务集”提交给底层的任务调度器（TaskScheduler）进行处理；Executor 向 SparkContext 申请任务，任务调度器将任务分发给 Executor 运行，同时，SparkContext 将应用程序代码发放给 Executor；

(4) 任务在 Executor 上运行，把执行结果反馈给任务调度器，然后反馈给 DAG 调度器，运行完毕后写入数据并释放所有资源。



图三

2. Spark 的四大特性

1、高效性

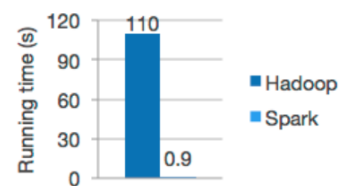
运行速度提高 100 倍。

Apache Spark 使用最先进的 DAG 调度程序，查询优化程序和物理执行引擎，实现批量和流式数据的高性能。

Speed

Run workloads 100x faster.

Apache Spark achieves high performance for both batch and streaming data, using a state-of-the-art DAG scheduler, a query optimizer, and a physical execution engine.



Logistic regression in Hadoop and Spark

2、易用性

Spark 支持 Java、Python 和 Scala 的 API，还支持超过 80 种高级算法，使用户可以快速构建不同的应用。而且 Spark 支持交互式的 Python 和 Scala 的 shell，可以非常方便地在这些 shell 中使用 Spark 集群来验证解决问题的方法。

Ease of Use

Write applications quickly in Java, Scala, Python, R, and SQL.

Spark offers over 80 high-level operators that make it easy to build parallel apps. And you can use it *interactively* from the Scala, Python, R, and SQL shells.

```
df = spark.read.json("logs.json")
df.where("age > 21")
.select("name.first").show()
```

Spark's Python DataFrame API
Read JSON files with automatic schema inference

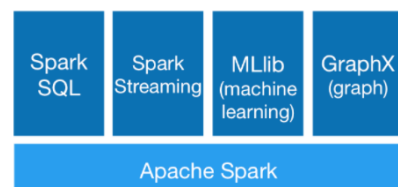
3、通用性

Spark 提供了统一的解决方案。Spark 可以用于批处理、交互式查询（Spark SQL）、实时流处理（Spark Streaming）、机器学习（Spark MLlib）和图计算（GraphX）。这些不同类型的处理都可以在同一个应用中无缝使用。Spark 统一的解决方案非常具有吸引力，毕竟任何公司都想用统一的平台去处理遇到的问题，减少开发和维护的人力成本和部署平台的物力成本。

Generality

Combine SQL, streaming, and complex analytics.

Spark powers a stack of libraries including [SQL and DataFrames](#), [MLlib](#) for machine learning, [GraphX](#), and [Spark Streaming](#). You can combine these libraries seamlessly in the same application.



4、兼容性

Spark 可以非常方便地与其他的开源产品进行融合。比如，Spark 可以使用 Hadoop 的 YARN 和 Apache Mesos 作为它的资源管理和调度器，并且可以处理所有 Hadoop 支持的数据，包括 HDFS、HBase 和 Cassandra 等。这对于已经部署 Hadoop 集群的用户特别重要，因为不需要做任何数据迁移就可以使用 Spark 的强大处理能力。Spark 也可以不依赖于第三方的资源管理和调度器，它实现了 Standalone 作为其内置的资源管理和调度框架，这样进一步降低了 Spark 的使用门槛，使得所有人都可以非常容易地部署和使用 Spark。此外，Spark 还提供了在 EC2 上部署 Standalone 的 Spark 集群的工具。

Runs Everywhere

Spark runs on Hadoop, Apache Mesos, Kubernetes, standalone, or in the cloud. It can access diverse data sources.

You can run Spark using its [standalone cluster mode](#), on [EC2](#), on [Hadoop YARN](#), on [Mesos](#), or on [Kubernetes](#). Access data in [HDFS](#), [Apache Cassandra](#), [Apache HBase](#), [Apache Hive](#), and hundreds of other data sources.



3. Spark 的组成

Spark 组成 (BDAS)：全称伯克利数据分析栈，通过大规模集成算法、机器、人之间展现大数据应用的一个平台。也是处理大数据、云计算、通信的技术解决方案。

它的主要组件有：

SparkCore：将分布式数据抽象为弹性分布式数据集（RDD），实现了应用任务调度、RPC、序列化和压缩，并为运行在其上的上层组件提供 API。

SparkSQL：Spark Sql 是 Spark 来操作结构化数据的程序包，可以让我使用 SQL 语句的方式来查询数据，Spark 支持 多种数据源，包含 Hive 表，parquest 以及 JSON 等内容。

SparkStreaming：是 Spark 提供的实时数据进行流式计算的组件。

MLlib：提供常用机器学习算法的实现库。

GraphX：提供一个分布式图计算框架，能高效进行图计算。

BlinkDB：用于在海量数据上进行交互式 SQL 的近似查询引擎。

Tachyon：以内存为中心高容错的的分布式文件系统。

4. 安装 scala

4.1 下载地址（本教程下载为 2.10.7）

```
>>wget http://www.scala-lang.org/download/
```

4.2 解压安装包：

```
>>mv scala-2.10.7.tgz ~/app
>>tar -zxvf scala-2.10.7.tgz
```

4.3 编辑.bash_profile 文件:

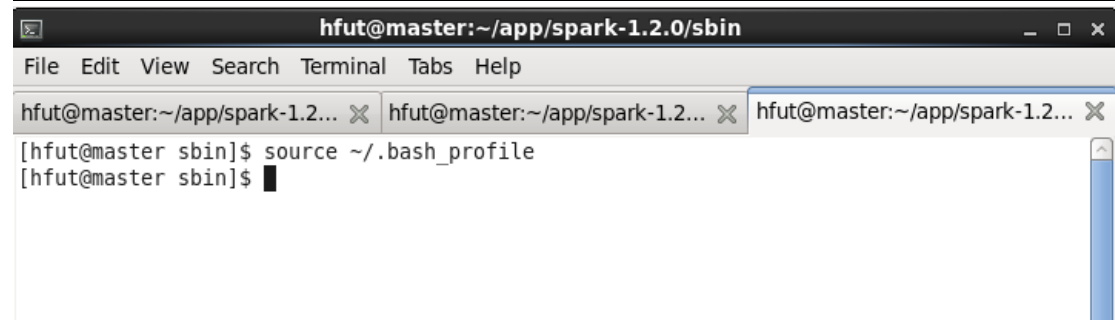
```
>>gedit ~/.bash_profile
```

4.4 在.bash_profile 文件中添加:

```
#Scala  
export SCALA_HOME=/home/hfut/app/scala-2.10.7  
export  
PATH=$JAVA_HOME/bin:$HADOOP_HOME/bin:$HIVE_HOME/bin:$SCALA_  
HOME/bin:$PATH
```

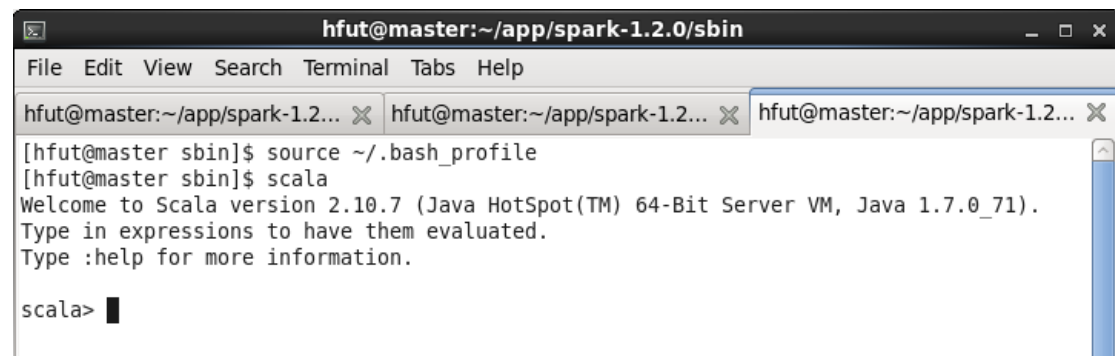
4.5 使环境变量生效

```
source ~/.bash_profile
```



A terminal window titled 'hfut@master:~/app/spark-1.2.0/sbin' with a menu bar (File, Edit, View, Search, Terminal, Tabs, Help). It shows three tabs for the same directory. The active tab displays the command '[hfut@master sbin]\$ source ~/.bash_profile' and the prompt '[hfut@master sbin]\$'.

4.6 验证 scala 安装:



A terminal window titled 'hfut@master:~/app/spark-1.2.0/sbin' with a menu bar (File, Edit, View, Search, Terminal, Tabs, Help). It shows three tabs for the same directory. The active tab displays the command '[hfut@master sbin]\$ source ~/.bash_profile', '[hfut@master sbin]\$ scala', and the output: 'Welcome to Scala version 2.10.7 (Java HotSpot(TM) 64-Bit Server VM, Java 1.7.0_71). Type in expressions to have them evaluated. Type :help for more information.' The prompt 'scala>' is shown.

5. 安装 spark (本文档使用的是 spark-1.2.0 版本)

5.1 下载安装包

```
>>wget http://d3kbcqa49mib13.cloudfront.net/spark-1.2.0-bin-hadoop1.tgz
```

5.2 解压安装包:

```
>>mv spark-1.2.0-bin-hadoop1.tgz ~/app  
>>cd ~/app  
>>tar -zxvf spark-1.2.0-bin-hadoop1.tgz
```



```
hfut@master:~/app
File Edit View Search Terminal Help
spark-1.2.0/examples/src/main/resources/full_user.avsc
spark-1.2.0/examples/src/main/resources/kv1.txt
spark-1.2.0/examples/scala-2.10/
spark-1.2.0/examples/scala-2.10/src/
spark-1.2.0/examples/scala-2.10/src/main/
spark-1.2.0/examples/scala-2.10/src/main/scala/
spark-1.2.0/examples/scala-2.10/src/main/scala/org/
spark-1.2.0/examples/scala-2.10/src/main/scala/org/apache/
spark-1.2.0/examples/scala-2.10/src/main/scala/org/apache/spark/
spark-1.2.0/examples/scala-2.10/src/main/scala/org/apache/spark/examples/
spark-1.2.0/examples/scala-2.10/src/main/scala/org/apache/spark/examples/streami
ng/
spark-1.2.0/examples/scala-2.10/src/main/scala/org/apache/spark/examples/streami
ng/KafkaWordCount.scala
spark-1.2.0/examples/scala-2.10/src/main/java/
spark-1.2.0/examples/scala-2.10/src/main/java/org/
spark-1.2.0/examples/scala-2.10/src/main/java/org/apache/
spark-1.2.0/examples/scala-2.10/src/main/java/org/apache/spark/
spark-1.2.0/examples/scala-2.10/src/main/java/org/apache/spark/examples/
spark-1.2.0/examples/scala-2.10/src/main/java/org/apache/spark/examples/streami
g/
spark-1.2.0/examples/scala-2.10/src/main/java/org/apache/spark/examples/streami
g/JavaKafkaWordCount.java
[hfut@master app]$
```

5.3 配置 spark 环境:

```
>>cd /spark-1.2.0-bin-hadoop1/conf/
>>cp spark-env.sh.template spark-env.sh
```

5.4 查看原始文件列表:

```
[hfut@master conf]$ ls -l
total 28
-rw-rw-r--. 1 hfut hfut 303 Dec 10 2014 fairscheduler.xml.template
-rw-rw-r--. 1 hfut hfut 620 Dec 10 2014 log4j.properties.template
-rw-rw-r--. 1 hfut hfut 5308 Dec 10 2014 metrics.properties.template
-rw-rw-r--. 1 hfut hfut 80 Dec 10 2014 slaves.template
-rw-rw-r--. 1 hfut hfut 507 Dec 10 2014 spark-defaults.conf.template
-rwxrwxr-x. 1 hfut hfut 3212 Dec 10 2014 spark-env.sh.template
[hfut@master conf]$
```

5.5 查看复制之后的文件列表:

```
[hfut@master conf]$ ls -l
total 32
-rw-rw-r--. 1 hfut hfut 303 Dec 10 2014 fairscheduler.xml.template
-rw-rw-r--. 1 hfut hfut 620 Dec 10 2014 log4j.properties.template
-rw-rw-r--. 1 hfut hfut 5308 Dec 10 2014 metrics.properties.template
-rw-rw-r--. 1 hfut hfut 80 Dec 10 2014 slaves.template
-rw-rw-r--. 1 hfut hfut 507 Dec 10 2014 spark-defaults.conf.template
-rwxrwxr-x. 1 hfut hfut 3212 May 7 19:22 spark-env.sh
-rwxrwxr-x. 1 hfut hfut 3212 Dec 10 2014 spark-env.sh.template
[hfut@master conf]$
```

5.6 将文件夹 `spark-1.2.0-bin-hadoop1` 改名为 `spark-1.2.0` (开始配置了 `spark-1.2.0` 包的环境但是失败, 后来重新下载了 `-bin-hadoop1` 的包, 环境就按照原先的配置了)。

5.7 编辑 spark-env.sh 文件：

```
>>gedit spark-env.sh
```

5.8 在 spark-env.sh 中添加下内容：

（其中将 JAVA_HOME、SCALA_HOME、HADOOP_CONF_DIR 以此改为 java 安装目录，scala 安装目录以及\$HADOOP_HOME/etc/hadoop）

```
export JAVA_HOME=/usr/java/jdk1.7.0_71
export SCALA_HOME=/home/hfut/app/scala-2.10.7
export HADOOP_CONF_DIR=/home/hfut/hadoop-2.5.2/etc/hadoop
```

5.9 配置 slaves：

```
>>cp slaves.templsate slaves
```

5.10 编辑 slave 文件：

```
>>gedit slaves
```

5.11 在 slaves 中将 localhost 改为：（其中 master 和 slave 均为主机名）

```
master
slave
```

5.12 将 .bash_profile 文件传到 slave 上：

```
>>scp ~/.bash_profile hfut@192.168.81.139:~/.bash_profile
```

5.13 将 spark 拷贝到 slave：（由于已经设置了免密登录，因此此处并不需要登陆密码，不然请检查免密登录）

```
>>scp ~/app/spark-1.2.0-bin-hadoop1
hfut@192.168.81.139:~/.bash_profile hfut@192.168.81.139:~/app/
```

6. 验证 spark 安装

6.1 启动 spark：

```
>>cd ~/app/spark-1.2.0/conf/sbin
>>start-all.sh
```

```
hfut@master:~/app/spark-1.2.0/sbin
File Edit View Search Terminal Tabs Help
hfut@master:~/app/spark-1.2.0/sbin hfut@master:~/app/spark-1.2.0/sbin
rst.
[hfut@master sbin]$ ./start-all.sh
org.apache.spark.deploy.master.Master running as process 3820. Stop it first.
slave: org.apache.spark.deploy.worker.Worker running as process 2889. Stop it fi
rst.
[hfut@master sbin]$ ./stop-all.sh
slave: stopping org.apache.spark.deploy.worker.Worker
master: no org.apache.spark.deploy.worker.Worker to stop
stopping org.apache.spark.deploy.master.Master
[hfut@master sbin]$ ./start-all.sh
starting org.apache.spark.deploy.master.Master, logging to /home/hfut/app/spark-1.2.0/s
bin/./logs/spark-hfut-org.apache.spark.deploy.master.Master-1-master.out
slave: starting org.apache.spark.deploy.worker.Worker, logging to /home/hfut/app/spark-
1.2.0/sbin/./logs/spark-hfut-org.apache.spark.deploy.worker.Worker-1-slave.out
master: starting org.apache.spark.deploy.worker.Worker, logging to /home/hfut/app/spark
-1.2.0/sbin/./logs/spark-hfut-org.apache.spark.deploy.worker.Worker-1-master.out
```

6.2 查看启动结果:

在 master 端看到 Master 和 Worker 说明启动成功,此处是已经启动了 hadoop 因此还有其他的出现。

```
Master>>jps

[hfut@master sbin]$ jps
4420 Master
2744 NameNode
3070 ResourceManager
2926 SecondaryNameNode
4579 Worker
4686 Jps
[hfut@master sbin]$
```

```
Slave>>jps

hfut@slave:~/Desktop
File Edit View Search Terminal Help
[hfut@slave Desktop]$ jps
2287 DataNode
2779 Jps
2164 Worker
2392 NodeManager
```

打开火狐浏览器,输入网址 `spark://master:7070` 查看启动情况:

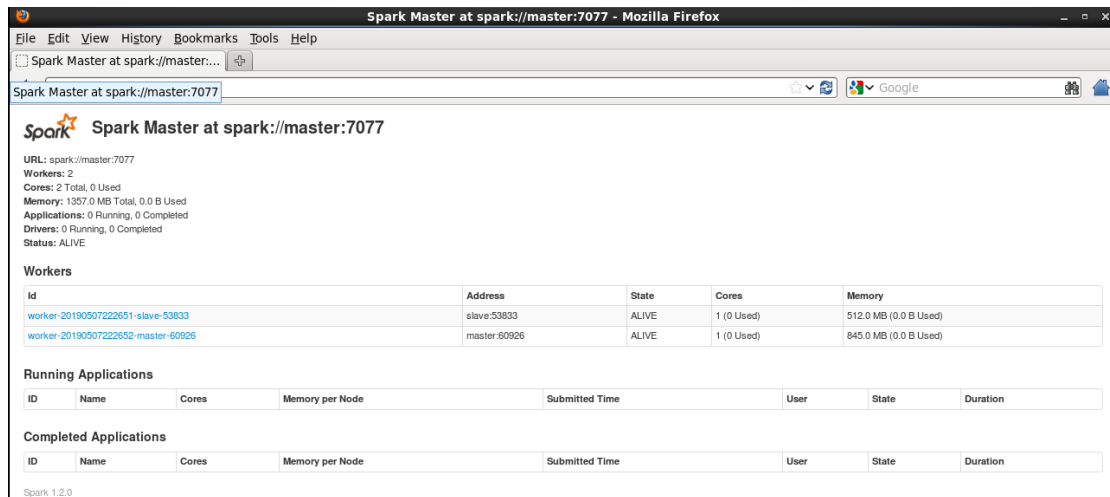
其他端口:

namenode 的 webUI 端口: 50070

yarn 的 web 端口: 8088

spark 集群的 web 端口: 8080

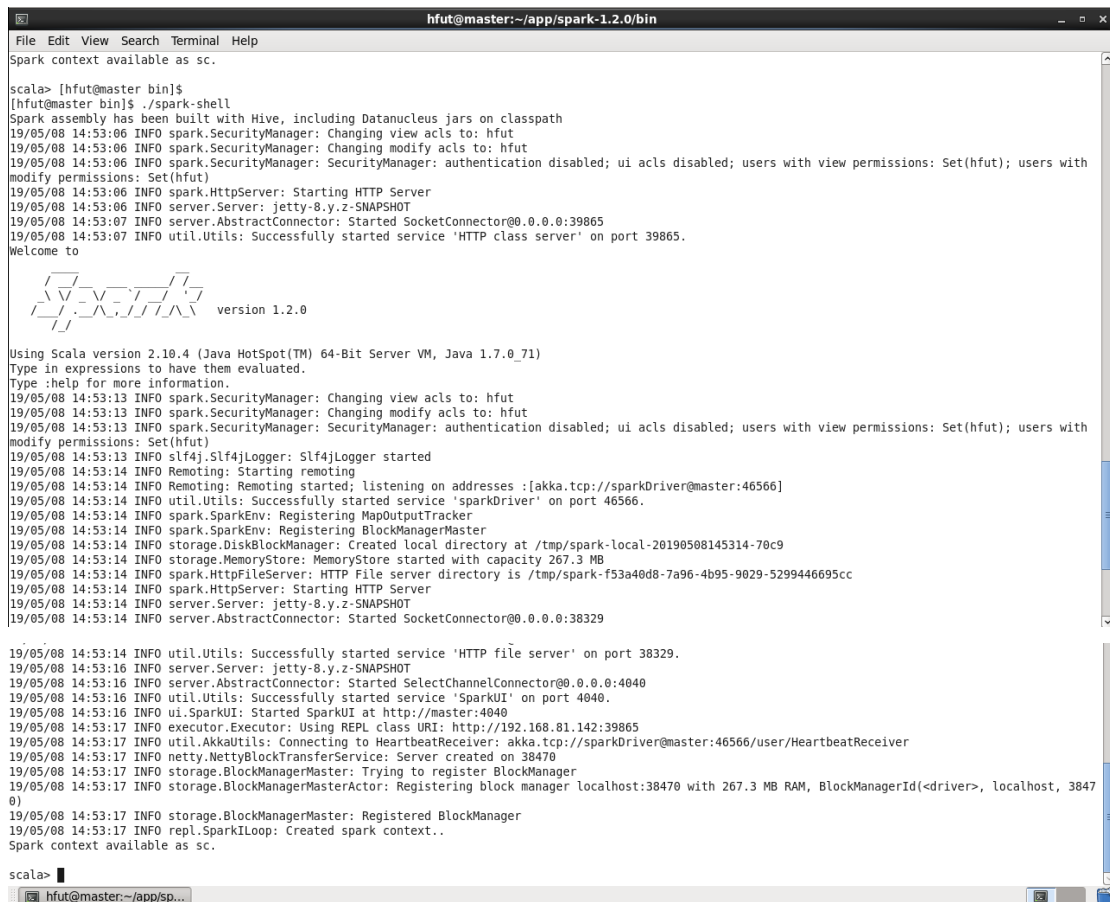
spark-job 监控端口: 4040



6.3 打开 spark-shell:

```
>>cd ~/app/spark-1.2.0/bin  
>>./spark-shell
```

出现如下结果说明安装成功:(此处建议第二行的命令改为:../bin/spark-shell, 因为上一步的命令可能会在后面执行 scala 语句的时候出错, 原因未知)



7. Spark-shell 的简单使用

7.1 入口: SQLContext (Starting Point: SQLContext)

Spark SQL 程序的主入口是 `SQLContext` 类或它的子类。创建一个基本的 `SQLContext`，你只需要 `SparkContext`，创建代码示例如下：

```
val sqlContext = new org.apache.spark.sql.SQLContext(sc)
```

```
scala> val sqlContext = new org.apache.spark.sql.SQLContext(sc)
sqlContext: org.apache.spark.sql.SQLContext = org.apache.spark.sql.SQLContext@386978c4
scala> █
```

除了基本的 `SQLContext`，也可以创建 `HiveContext`。`SQLContext` 和 `HiveContext` 区别与联系为：

`SQLContext` 现在只支持 SQL 语法解析器（SQL-92 语法）

`HiveContext` 现在支持 SQL 语法解析器和 HiveSQL 语法解析器，默认为 HiveSQL 语法解析器，用户可以通过配置切换到 SQL 语法解析器，来运行 HiveSQL 不支持的语法。

使用 `HiveContext` 可以使用 Hive 的 UDF，读写 Hive 表数据等 Hive 操作。`SQLContext` 不可以对 Hive 进行操作。

Spark SQL 未来的版本会不断丰富 `SQLContext` 的功能，做到 `SQLContext` 和 `HiveContext` 的功能容和，最终可能两者会统一成一个 `Context`

`HiveContext` 包装了 Hive 的依赖包，把 `HiveContext` 单独拿出来，可以在部署基本的 Spark 的时候就不需要 Hive 的依赖包，需要使用 `HiveContext` 时再把 Hive 的各种依赖包加进来。

SQL 的解析器可以通过配置 `spark.sql.dialect` 参数进行配置。在 `SQLContext` 中只能使用 Spark SQL 提供的”sql“解析器。在 `HiveContext` 中默认解析器为”hiveql“，也支持”sql“解析器。

7.2 RDD 创建：

```
val intRDD = sc.sparkContext.parallelize(List(3,1,2,5,5))
print("打印 intRDD 的结果: "+intRDD.collect().mkString(", "))
println("*****单个 RDD 转换运算*****")

def addOne(x:Int):Int =
{
    return (x+1)
}
```

```
//map 运算：对 RDD 中每一个元素做一个转换操作，生成一个新的 RDD

println(" 使用 具体的 函数 完成 map 运算 :
"+intRDD.map(addOne).collect().mkString(","))

println(" 使用 匿名 函数 完成 map 运算 :
"+intRDD.map(x=>x+1).collect().mkString(","))

println(" 使用 匿名 函数 和 匿名 参数 完成 map 运算 :
"+intRDD.map(_+1).collect().mkString(","))

println(" 使用 匿名 函数 完成 map 运算 :
"+stringRDD.map(x=>"fruit"+x).collect().mkString(","))

//filter 运算：对 RDD 中每一个元素进行筛选，生成一个新的 RDD

println("使用匿名函数筛选 intRDD 中小于 5 的元素："+intRDD.filter(x=>x <
5).collect().mkString(","))

println("使用匿名函数和匿名参数筛选 intRDD 中小于 5 的元素："+intRDD.filter(_
< 5).collect().mkString(","))

println(" 使用 匿名 函数 筛选 stringRDD 中 包含 ra 的 元素 :
"+stringRDD.filter(x=>x.contains('ra')).collect().mkString(","))

//distinct 运算：对 RDD 中元素进行去重

println("对 intRDD 元素进行去重"+intRDD.distinct().collect().mkString(","))
println("对 StringRDD 元素进行去重"+stringRDD.distinct().collect().mkString(","))

//randomSplit 运算，按照指定的比例将 RDD 进行划分

val sRDD = intRDD.randomSplit(Array(0.4,0.6))

println("分割后第一个 RDD 为："+sRDD(0).collect().mkString(","))
println("分割后第二个 RDD 为："+sRDD(1).collect().mkString(","))

//groupBy 运算：可以按照传入的匿名函数规则，将数据分为多个 Array,返回
```

```
Array[(String,Iterable[Int])]
```

```
val gRDD = intRDD.groupBy(x => {if(x%2==0) "even" else "odd"}).collect()

println("偶数数组为: "+gRDD(0))

println("奇数数组为: "+gRDD(1))
```

结果:

打印 intRDD 的结果: 3, 1, 2, 5, 5

*****单个 RDD 转换运算*****

使用具体的函数完成 map 运算: 4, 2, 3, 6, 6

使用匿名函数完成 map 运算: 4, 2, 3, 6, 6

使用匿名函数和匿名参数完成 map 运算: 4, 2, 3, 6, 6

使用匿名函数完成 map 运算:
fruitApple, fruitOrange, fruitBanana, fruitGrape, fruitApple

使用匿名函数筛选 intRDD 中小于 5 的元素: 3, 1, 2

使用匿名函数和匿名参数筛选 intRDD 中小于 5 的元素: 3, 1, 2

使用匿名函数筛选 stringRDD 中包含 ra 的元素:

对 intRDD 元素进行去重 2, 1, 3, 5

对 StringRDD 元素进行去重 Orange, Apple, Grape, Banana

分割后第一个 RDD 为: 1, 2, 5

分割后第二个 RDD 为: 3, 5

偶数数组为: (even, CompactBuffer(2))

奇数数组为: (odd, CompactBuffer(3, 1, 5, 5))

7.3 案例分析

该案例中, 我们将假设我们需要统计一个 1 万人口的所有人的平均年龄, 当然如果您想测试 Spark 对于大数据的处理能力, 您可以把人口数放的更大, 比如 1 亿人口, 当然这个取决于测试所用集群的存储容量。假设这些年龄信息都存储在一个文件里, 并且该文件的格式如下, 第一列是 ID, 第二列是年龄。

年龄生成代码:

```
FILE * fp;

fp = fopen("ageNum.txt", "w+");
```



```

for (int i = 0; i < 10000; i++) {
    srand((UINT)GetCurrentTime());
    fprintf(fp, "%d %d\n", i, rand()%100);
    printf("%d %d\n", i, rand() % 100);
}
fclose(fp);

```

```

1 0 45
2 1 37
3 2 29
4 3 15
5 4 52
6 5 63

```

要计算平均年龄，那么首先需要对源文件对应的 RDD 进行处理，也就是将它转化成一个只包含年龄信息的 RDD，其次是计算元素个数即为总人数，然后把所有年龄数加起来，最后平均年龄=总年龄/人数。

对于第一步我们需要使用 map 算子把源文件对应的 RDD 映射成一个新的只包含年龄数据的 RDD，很显然需要对在 map 算子的传入函数中使用 split 方法，得到数组后只取第二个元素即为年龄信息；第二步计算数据元素总数需要对于第一步映射的结果 RDD 使用 count 算子；第三步则是使用 reduce 算子对只包含年龄信息的 RDD 的所有元素用加法求和；最后使用除法计算平均年龄即可。由于本例输出结果很简单，所以只打印在控制台即可。

```
>>val textfile1 = sc.textFile("file:///home/hfut/ageNum.txt")
```

```

scala> val textfile1 = sc.textFile("file:///home/hfut/ageNum.txt")
19/05/09 15:09:57 INFO storage.MemoryStore: ensureFreeSpace(33442) called with curMem=0, maxMem=280
248975
19/05/09 15:09:57 INFO storage.MemoryStore: Block broadcast_0 stored as values in memory (estimated
size 32.7 KB, free 267.2 MB)
19/05/09 15:09:58 INFO storage.MemoryStore: ensureFreeSpace(5069) called with curMem=33442, maxMem=
280248975
19/05/09 15:09:58 INFO storage.MemoryStore: Block broadcast_0_piece0 stored as bytes in memory (est
imated size 5.0 KB, free 267.2 MB)
19/05/09 15:09:58 INFO storage.BlockManagerInfo: Added broadcast_0_piece0 in memory on localhost:42
104 (size: 5.0 KB, free: 267.3 MB)
19/05/09 15:09:58 INFO storage.BlockManagerMaster: Updated info of block broadcast_0_piece0
19/05/09 15:09:58 INFO spark.SparkContext: Created broadcast 0 from textFile at <console>:12
textfile1: org.apache.spark.rdd.RDD[String] = file:///home/hfut/ageNum.txt MappedRDD[1] at textFile
at <console>:12

```

```
>>val count = textfile1.count()
```



```

hfut@master:~/app/spark-1.2.0/sbin
hfut@master:~/Desktop

scala> val count = textfile1.count()
19/05/09 15:16:34 INFO spark.SparkContext: Starting job: count at <console>:14
19/05/09 15:16:34 INFO scheduler.DAGScheduler: Got job 1 (count at <console>:14) with 1 output partitions (allowLocal=false)
19/05/09 15:16:34 INFO scheduler.DAGScheduler: Final stage: Stage 1(count at <console>:14)
19/05/09 15:16:34 INFO scheduler.DAGScheduler: Parents of final stage: List()
19/05/09 15:16:34 INFO scheduler.DAGScheduler: Missing parents: List()
19/05/09 15:16:34 INFO scheduler.DAGScheduler: Submitting Stage 1 (file:///home/hfut/ageNum.txt MappedRDD[1] at textFile at <console>:12), which has no missing parents
19/05/09 15:16:34 INFO storage.MemoryStore: ensureFreeSpace(2528) called with curMem=42918, maxMem=280248975
19/05/09 15:16:34 INFO storage.MemoryStore: Block broadcast 2 stored as values in memory (estimated size 2.5 KB, free 267.2 MB)
19/05/09 15:16:34 INFO storage.MemoryStore: ensureFreeSpace(1879) called with curMem=45446, maxMem=280248975
19/05/09 15:16:34 INFO storage.MemoryStore: Block broadcast 2_piece0 stored as bytes in memory (estimated size 1879.0 B, free 267.2 MB)
19/05/09 15:16:34 INFO storage.BlockManagerInfo: Added broadcast 2_piece0 in memory on localhost:42104 (size: 1879.0 B, free: 267.3 MB)
19/05/09 15:16:34 INFO scheduler.DAGScheduler: Created broadcast 2 from broadcast at DAGScheduler.scala:838
19/05/09 15:16:34 INFO scheduler.DAGScheduler: Submitting 1 missing tasks from Stage 1 (file:///home/hfut/ageNum.txt MappedRDD[1] at textFile at <console>:12)
19/05/09 15:16:34 INFO scheduler.TaskSchedulerImpl: Adding task set 1.0 with 1 tasks
19/05/09 15:16:34 INFO scheduler.TaskSetManager: Starting task 0.0 in stage 1.0 (TID 1, localhost, PROCESS_LOCAL, 1291 bytes)
19/05/09 15:16:34 INFO executor.Executor: Running task 0.0 in stage 1.0 (TID 1)
19/05/09 15:16:34 INFO rdd.HadoopRDD: Input split: file:/home/hfut/ageNum.txt:0+87966
19/05/09 15:16:34 INFO executor.Executor: Finished task 0.0 in stage 1.0 (TID 1). 1757 bytes result sent to driver
19/05/09 15:16:34 INFO scheduler.DAGScheduler: Stage 1 (count at <console>:14) finished in 0.042 s
19/05/09 15:16:34 INFO scheduler.DAGScheduler: Job 1 finished: count at <console>:14, took 0.162399 s
count: Long = 10000

scala> 19/05/09 15:16:34 INFO scheduler.TaskSetManager: Finished task 0.0 in stage 1.0 (TID 1) in 43 ms on localhost (1/1)
19/05/09 15:16:34 INFO scheduler.TaskSchedulerImpl: Removed TaskSet 1.0, whose tasks have all completed, from pool

```

```
>>val ageData = textfile1.map(line => line.split(" ")(1))
```

```

scala> val ageData = textfile1.map(line => line.split(" ")(1))
ageData: org.apache.spark.rdd.RDD[String] = MappedRDD[2] at map at <console>:14

scala> █

```

```
>>val totalAge = ageData.map(age => Integer.parseInt
(String.valueOf(age))).collect().reduce((a,b) => a+b)
```

```

scala> val totalAge = ageData.map(age => Integer.parseInt(String.valueOf(age))).collect().reduce((a,b) => a+b)
19/05/09 15:17:57 INFO spark.SparkContext: Starting job: collect at <console>:16
19/05/09 15:17:57 INFO scheduler.DAGScheduler: Got job 2 (collect at <console>:16) with 1 output partitions (allowLocal=false)
19/05/09 15:17:57 INFO scheduler.DAGScheduler: Final stage: Stage 2(collect at <console>:16)
19/05/09 15:17:57 INFO scheduler.DAGScheduler: Parents of final stage: List()
19/05/09 15:17:57 INFO scheduler.DAGScheduler: Missing parents: List()
19/05/09 15:17:57 INFO scheduler.DAGScheduler: Submitting Stage 2 (MappedRDD[3] at map at <console>:16), which has no missing parents
19/05/09 15:17:57 INFO storage.MemoryStore: ensureFreeSpace(2896) called with curMem=47325, maxMem=280248975
19/05/09 15:17:57 INFO storage.MemoryStore: Block broadcast 3 stored as values in memory (estimated size 2.8 KB, free 267.2 MB)
19/05/09 15:17:57 INFO storage.MemoryStore: ensureFreeSpace(2084) called with curMem=50221, maxMem=280248975
19/05/09 15:17:57 INFO storage.MemoryStore: Block broadcast 3_piece0 stored as bytes in memory (estimated size 2.0 KB, free 267.2 MB)
19/05/09 15:17:57 INFO storage.BlockManagerInfo: Added broadcast 3_piece0 in memory on localhost:42104 (size: 2.0 KB, free: 267.3 MB)
19/05/09 15:17:57 INFO storage.BlockManagerMaster: Updated info of block broadcast 3_piece0
19/05/09 15:17:57 INFO spark.SparkContext: Created broadcast 3 from broadcast at DAGScheduler.scala:838
19/05/09 15:17:57 INFO scheduler.DAGScheduler: Submitting 1 missing tasks from Stage 2 (MappedRDD[3] at map at <console>:16)
19/05/09 15:17:57 INFO scheduler.TaskSchedulerImpl: Adding task set 2.0 with 1 tasks
19/05/09 15:17:57 INFO scheduler.TaskSetManager: Starting task 0.0 in stage 2.0 (TID 2, localhost, PROCESS_LOCAL, 1291 bytes)
19/05/09 15:17:57 INFO executor.Executor: Running task 0.0 in stage 2.0 (TID 2)
19/05/09 15:17:57 INFO rdd.HadoopRDD: Input split: file:/home/hfut/ageNum.txt:0+87966
19/05/09 15:17:57 INFO executor.Executor: Finished task 0.0 in stage 2.0 (TID 2). 41897 bytes result sent to driver
19/05/09 15:17:57 INFO scheduler.DAGScheduler: Stage 2 (collect at <console>:16) finished in 0.279 s
19/05/09 15:17:57 INFO scheduler.DAGScheduler: Job 2 finished: collect at <console>:16, took 0.322917 s
19/05/09 15:17:57 INFO scheduler.TaskSetManager: Finished task 0.0 in stage 2.0 (TID 2) in 276 ms on localhost (1/1)
19/05/09 15:17:57 INFO scheduler.TaskSchedulerImpl: Removed TaskSet 2.0, whose tasks have all completed, from pool
totalAge: Int = 493171

```

```
>>println("Total Age:" + totalAge + ";Number of People:" + count )
```

```

scala> println("Total Age:" + totalAge + ";Number of People:" + count )
Total Age:493171;Number of People:10000

scala> █

```

```
>>val avgAge : Double = totalAge.toDouble / count.toDouble
```

```

scala> val avgAge : Double = totalAge.toDouble / count.toDouble
avgAge: Double = 49.3171

```

```
>>println("Average Age is " + avgAge)
```

```

scala> println("Average Age is " + avgAge)
Average Age is 49.3171

scala> █

```

参考文献:

<http://dmlab.xmu.edu.cn/blog/1709-2/>

<https://www.w3cschool.cn/spark/>

<https://blog.csdn.net/u011630575/article/details/56556949>