1、Spark简介-概念

Spark是**基于内存计算**的大数据**分布式计算框架**。Spark基于内存计算，提高了在**大数据**环境下数据处理的实时性，同时保证了高容错性和高可伸缩性，允许用户将Spark部署在大量廉价硬件之上，形成集群。

主要特点：

分布式计算

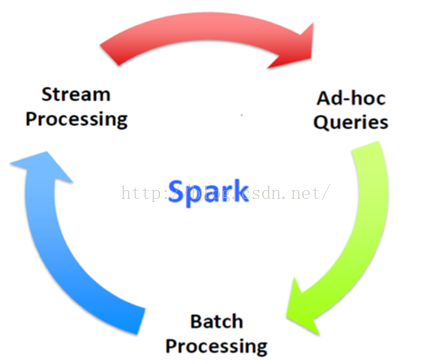
内存计算

容错

多计算范式

Spark于2009 年诞生于加州大学伯克利分销AMPLab。目前，已经成为Apache软件基金会旗下顶级开源项目。

在“**One Stack to rule them all**”思想的引领下，Spark成功的使用Spark SQL、Spark Streaming、MLLib、GraphX近乎完美的解决了大数据中Batch Processing、Streaming Processing、Ad-hoc Query等三大核心问题。



2、Spark简介-历史

2009年：Spark诞生于AMPLab

2010年：开源

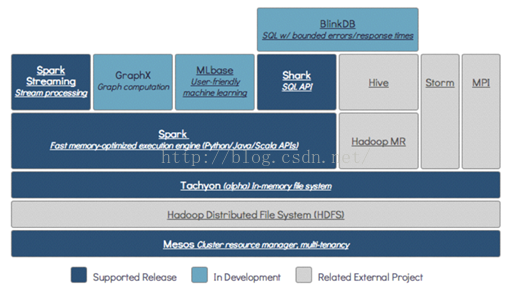
2013年6月：Apache孵化器项目

2014年2月：Apache顶级项目

Now：Contribututors>450人

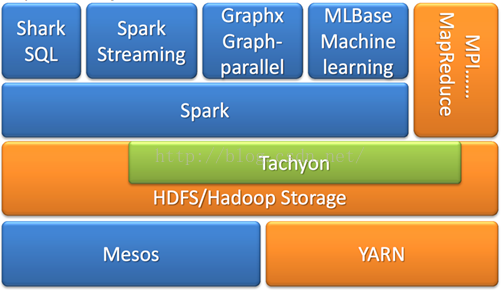
3、Spark简介-BDAS生态系统

3.1 BDAS(the Berkeley Data Analytics Stack) 伯克利数据分析栈



3.2 Spark Ecosystem

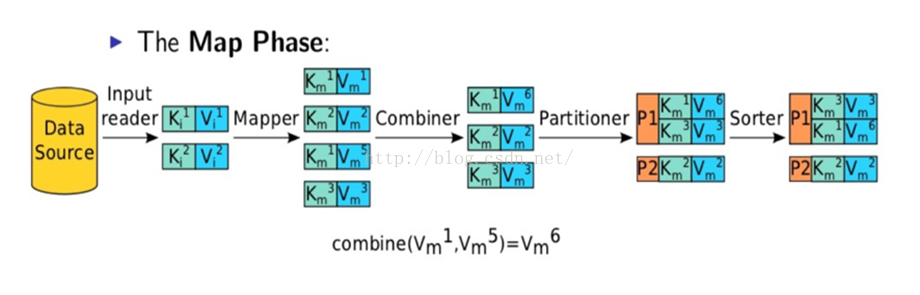
Spark成功的使用Spark SQL、Spark Streaming、MLLib、GraphX近乎完美的解决了大数据中Batch Processing、Streaming Processing、Ad-hoc Query等三大核心问题，更为美妙的是在Spark中Spark SQL、Spark Streaming、MLLib、GraphX四大子框架和库之间可以无缝的共享数据和操作。



[**002-mapreduce经典模型**](http://blog.csdn.net/shenfuli/article/details/48162695)

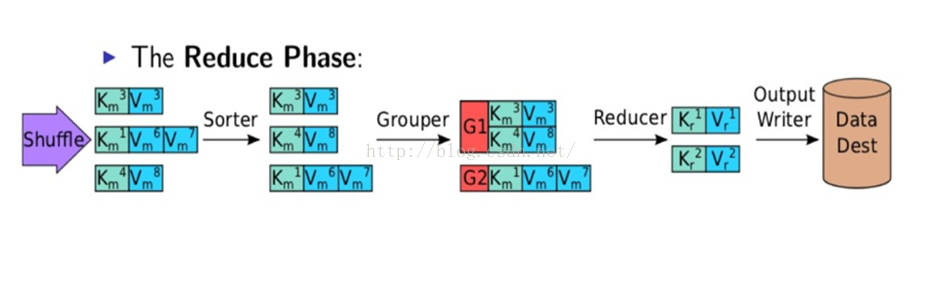
1、MapReduce Map

Map(k1,v1)->List(k2,v2)

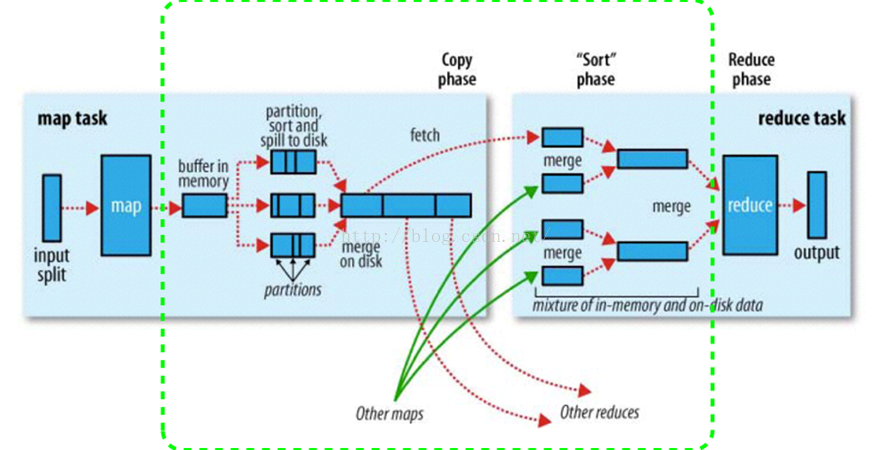


2、MapReduce Reduce

Reduce(k2,List(v2))->List(k3,v3)



3、Shuffle 



[**003-spark名词和架构介绍**](http://blog.csdn.net/shenfuli/article/details/48162729)

1、Spark名词

1、RDD： 数据集合

2、Application： Spark用户程序

3、Driver Program： 运行main函数并且新建SparkContext的程序

4、Cluster Manager ： Spark 集群资源调度服务（standlone，mesos，yarn）

5、Executor: worker node 的一个进程，负责运行任务

6、Task： 被送到某个executor上的工作单元

7、job： 包含多个RDD一级作用于RDD上的各种operate

8、Stage： 一个job分成多个节点

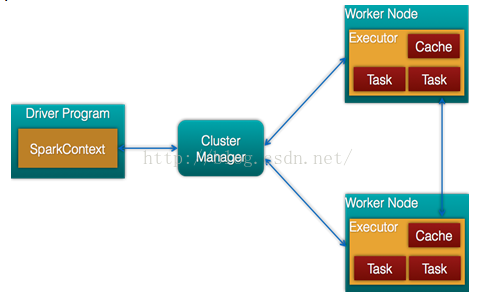
9、Narrow Dependency

10、Wide Dependency ： 宽依赖

11、Caching Managerment ： 缓存管理

2、Spark 集群架构

Master负责集群整体资源管理和调度，Worker负责单个节点的资源管理，Driver程序是应用逻辑执行的起点，多个Executor用来对数据进行并行处理。



Spark架构的构成

（1） Cluster Manager：　在standalone模式中即为Master主节点，控制整个集群，监控worker。在YARN模式中为资源管理器。

（2） Worker节点：从节点，负责控制计算节点，启动Executor或者Driver。在YARN模式中为NodeManager，负责计算节点控制。

（3） Driver： 运行Application 的main()函数

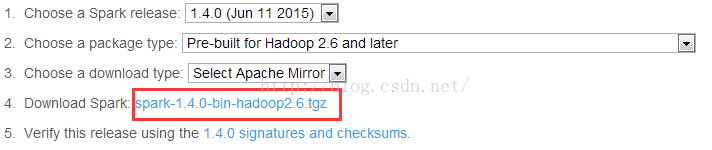
（4） Executor：执行器，是为某个Application运行在worker node上的一个进程，启动线城市运行任务上。每个Application拥有独立的一组executors。

（5）SparkContext： 整个应用的上下文，控制应用的生命周期。

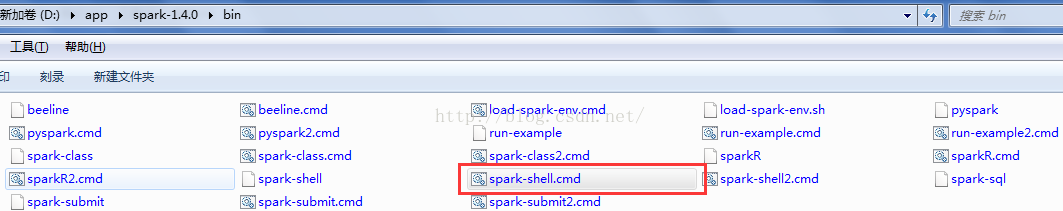
spark local 模式  在window或者linux下都一样，都是启动一个节点，只能用于测试。

1、下载Spark

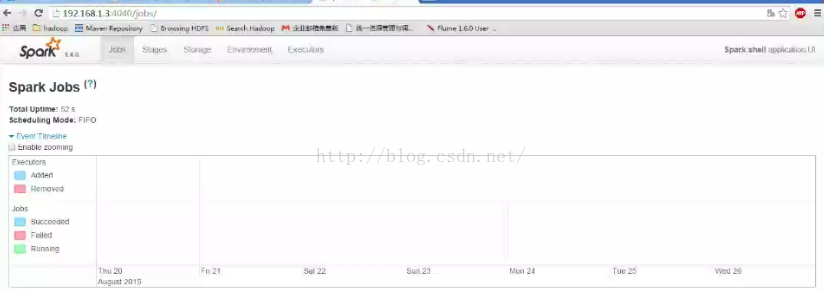
官方地址：http://spark.apache.org/downloads.html



2、window 7 下解压，然后点击“”启动spark



3、spark验证是否启动成功

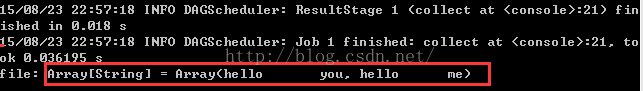


4、spark计算验证

4.1 加载文本信息

加载一个或者多个文件，使用spark需要上下文对象sc

scala> var file=sc.textFile("d:/test/a.txt").collect



4.2 可以使用-master启动多个线程

http://img.blog.csdn.net/20150901201003611?watermark/2/text/aHR0cDovL2Jsb2cuY3Nkbi5uZXQv/font/5a6L5L2T/fontsize/400/fill/I0JBQkFCMA==/dissolve/70/gravity/Center

4.3 单词统计

scala> sc.textFile("d:/test/a.txt").flatMap( line=>line.split("\\t") ).map( word=>(word,1) ).reduceByKey(\_ + \_).collect

http://img.blog.csdn.net/20150901200948262?watermark/2/text/aHR0cDovL2Jsb2cuY3Nkbi5uZXQv/font/5a6L5L2T/fontsize/400/fill/I0JBQkFCMA==/dissolve/70/gravity/Center

[**005-spark standalone模式安装**](http://blog.csdn.net/shenfuli/article/details/48162805)

spark standalone模式不同于单节点本地模式，它有主从节点,分别为Master和worker

Spark standlone规划

Master节点：  192.168.2.20

Worker节点： 192.168.2.20,192.168.2.33

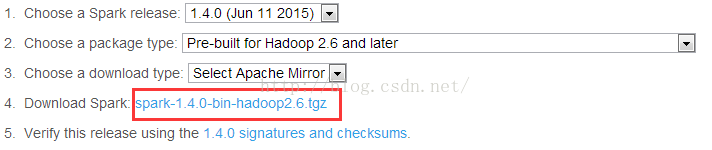
其中： 地址和域名的映射关系为

192.168.2.20 mycluster

192.168.2.33 cloud03

1、spark 下载

官方地址：http://spark.apache.org/downloads.html



2、spark安装

2.1 解压spark

[hadoop@mycluster ~]$ tar -zxvf spark-1.4.0-bin-hadoop2.6.tgz -C app/

2.2 设置主节点地址和JAVA\_HOME变量

vi spark-env.sh

SPARK\_MASTER\_IP=192.168.2.20

export JAVA\_HOME=/home/hadoop/app/jdk1.7.3

2.3 设置从节点地址

vi slaves

192.168.2.20

192.168.2.33

2.4 拷贝spark的目录到其他节点上

备注： 20节点和3节点需要建立ssh免登录。

3、spark standalone启动

在主节点上启动spark standlone模式，执行下面命令

cd  $SPARK\_HOME/sbin

./start-all.sh

3.1 8080查看master的工作状态

http://192.168.2.20:8080/

3.2 8081查看worker的工作状态

<http://192.168.2.20:8081/>

3.3 通过jps可以查看主从节点是否启动

[hadoop@mycluster sbin]$ jps  
2822 Master  
3452 Jps  
3354 Worker

[hadoop@cloud03 spark-1.4.0-bin-hadoop2.6]$ jps  
2306 Worker  
2403 Jps

4、启动spark shell终端

启动成功后，通过4040端口查看job列表和状态，即http://192.168.2.20:4040

5、通过shell下达命令

在本地建立测试文件

[hadoop@mycluster ~]$ cat /home/hadoop/wc.txt  
hello   me  
hello   you  
hello   china  
hello   you

5.1加载数据文件，可以是本地路径，也是是HDFS路径或者其它

scala> var textFile = sc.textFile("/home/hadoop/wc.txt");

5.2 列出文件行数

scala> textFile.count();

5.3 列出首行内容

scala> textFile.first()

5.4 过滤

textFile.filter(line => line.contains("me"))

[**007-spark的wordCount**](http://blog.csdn.net/shenfuli/article/details/48162827)

测试文本内容

[hadoop@mycluster ~]$ cat /home/hadoop/wc.txt  
hello   me  
hello   you  
hello   china  
hello   you

1、读取本地或者HDFS文件

spark启动时候生成SparkContext 对象sc，通过spark的上下文对象sc读取文件

命令：scala> var textFile = sc.textFile("/home/hadoop/wc.txt").collect

执行结果：textFile: Array[String] = Array(hello   me, hello       you, hello      china, hello    you)

2、执行文件

2.1 flatMap 对读取的结果通过制表符方式平摊

scala> var textFile = sc.textFile("/home/hadoop/wc.txt").flatMap(line => line.split("\t")).collect

或者

scala> var textFile = sc.textFile("/home/hadoop/wc.txt").flatMap(\_.split("\t")).collect

结果：

textFile: Array[String] = Array(hello, me, hello, you, hello, china, hello, you)

2.2  map(word=>(word,1))   word表示每个单词，每个单词为1

scala> var textFile = sc.textFile("/home/hadoop/wc.txt").flatMap(line => line.split("\t")).map(word => (word,1)).collect

或者

scala> var textFile = sc.textFile("/home/hadoop/wc.txt").flatMap(\_.split("\t")).map((\_,1)).collect

结果：

textFile: Array[(String, Int)] = Array((hello,1), (me,1), (hello,1), (you,1), (hello,1), (china,1), (hello,1), (you,1))

2.3 执行reduceByKey函数

scala> var textFile = sc.textFile("/home/hadoop/wc.txt").flatMap(\_.split("\t")).map((\_,1)).reduceByKey( (a,b) => a + b ).collect

或者

var textFile = sc.textFile("/home/hadoop/wc.txt").flatMap(\_.split("\t")).map((\_,1)).reduceByKey( \_ + \_ ).collect

结果：

textFile: Array[(String, Int)] = Array((hello,4), (me,1), (you,2), (china,1))

2.4 key 字段进行排序

scala> var textFile = sc.textFile("/home/hadoop/wc.txt").flatMap(\_.split("\t")).map((\_,1)).reduceByKey( \_ + \_ ).sortByKey(true).collect

结果：

textFile: Array[(String, Int)] = Array((china,4), ("hello ",1), (me,1), (you,2))

2.5 输出结果保存本地或者HDFS上

var textFile = sc.textFile("/home/hadoop/wc.txt").flatMap(\_.split("\t")).map((\_,1)).reduceByKey( \_ + \_ ).sortByKey(true).saveAsTextFile("/home/hadoop/output")

执行结果：

[hadoop@mycluster output]$ more part-00000  
(hello,4)  
(me,1)  
[hadoop@mycluster output]$ more part-00001  
(you,2)  
(china,1)

2.6 让输出结果仅生成一个文件

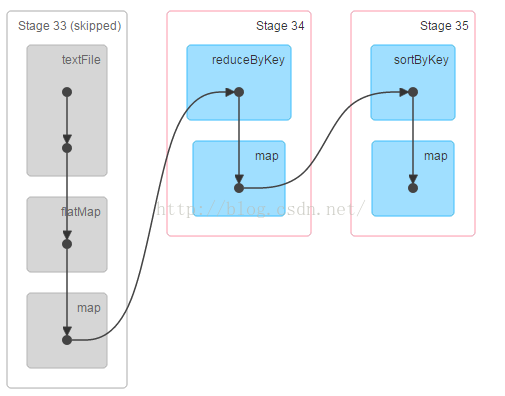
var textFile = sc.textFile("/home/hadoop/wc.txt").flatMap(\_.split("\t")).map((\_,1)).reduceByKey(\_+\_).repartition(1).saveAsTextFile("/home/hadoop/output")

执行结果：

[hadoop@mycluster output]$ more part-00000  
(hello,4)  
(me,1)  
(you,2)  
(china,1)

2.6  出现次数最多的单词排在前面

var textFile = sc.textFile("hdfs://mycluster:9000/wc.txt").flatMap(\_.split("\t")).map((\_,1)).reduceByKey(\_+\_).map(x=>(x.\_2,x.\_1)).sortByKey(false).map( x => (x.\_2,x.\_1) ).collect



结果： textFile: Array[(String, Int)] = Array((hello,4), (you,2), (me,1), (china,1))

备注：

以上就是wordcount的例子。下面给出读取hdfs上的数据的案例

var textFile = sc.textFile("hdfs://mycluster:9000/wc.txt").flatMap(\_.split("\t")).map((\_,1)).reduceByKey(\_+\_).repartition(1).saveAsTextFile("hdfs://mycluster:9000/output")

执行结果：

[hadoop@mycluster output]$ hdfs dfs -cat hdfs://mycluster:9000/output/part-00000  
(hello,4)  
(me,1)  
(you,2)  
(china,1)

[**008-spark 的调度**](http://blog.csdn.net/shenfuli/article/details/48162851)

系统的设计很重要的一个环节就是资源调度。设计者将资源进行不同粒度的抽象建模，然后将资源统一放入调度器，通过一定的算法进行调度，最终要达到高吞吐或者低延迟的目的。

Application调度就是由用户提交到Spark中的作业集合，通过一定的算法，对每个按照一定次序分配集群中资源的过程。

例如：FIFO模式，用户先提交的作业1优先分配需要的资源，之后提交的作业2再分配资源，以此类推。

Application调度模式:

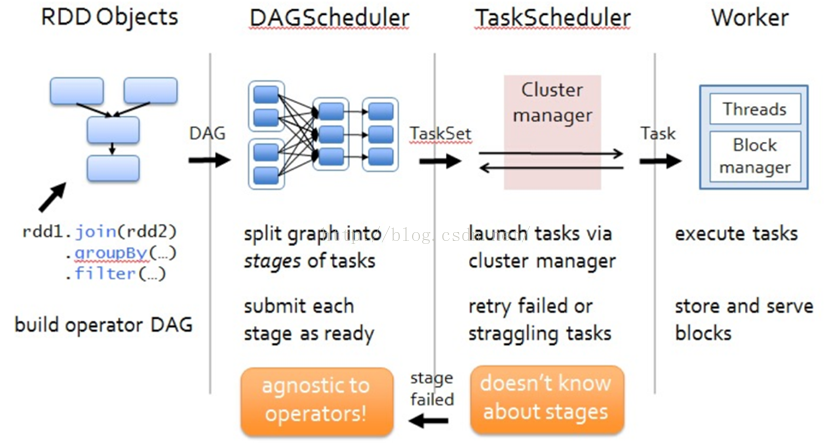
（1）Standalone 模式： FIFO模式

（2）Mesos：粗粒度模式和细粒度模式

（3）YARN：独占模式

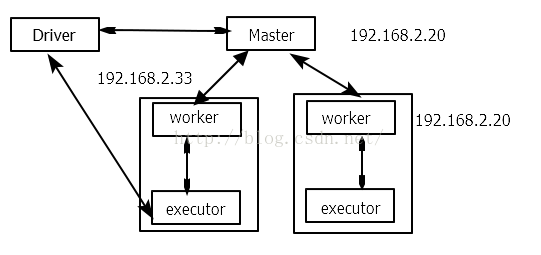
Job调度

Job调度就是在Application一组Job集合，在Application分配到的资源量下，通过一定的算法，对每个按一定次序分配的Application中资源的过程。



[**010-spark standalone模式Scala版本WordCount代码**](http://blog.csdn.net/shenfuli/article/details/48162871)

1、服务器部署架构



/home/hadoop/app/spark/conf/spark-env.sh 的配置文件：

#!/usr/bin/env bash  
SPARK\_MASTER\_IP=mycluster  
export JAVA\_HOME=/home/hadoop/app/jdk1.7.0\_76  
export SCALA\_HOME=/home/hadoop/app/scala  
export HADOOP\_HOME=/home/hadoop/app/hadoop-2.6.0  
export HIVE\_HOME=/home/hadoop/app/hive  
export SPARK\_CLASSPATH=$HIVE\_HOME/lib/mysql-connector-java-5.1.28.jar

slaves配置文件的内容  
# A Spark Worker will be started on each of the machines listed below.  
192.168.2.20  
192.168.2.33

2、统计单词Scala程序

2.1 引入Spark对应的jar包，添加pom.xml 中

**[html]** [view plain](http://blog.csdn.net/shenfuli/article/details/48162871) [copy](http://blog.csdn.net/shenfuli/article/details/48162871)

[print?](http://blog.csdn.net/shenfuli/article/details/48162871)

1. **<?xml** version= "1.0" encoding ="UTF-8"**?>**
2. **<project** xmlns= "http://maven.apache.org/POM/4.0.0"
3. xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4. xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd" **>**
5. **<modelVersion** **>**4.0.0 **</modelVersion>**
7. **<groupId** **>**testSpark **</groupId>**
8. **<artifactId** **>**testSpark **</artifactId>**
9. **<version** **>**1.0-SNAPSHOT **</version>**
11. **<repositories** **>**
12. **<repository>**
13. **<id>** Akka repository **</id>**
14. **<url>** http://repo.akka.io/releases**</url** **>**
15. **</repository>**
16. **<repository>**
17. **<id>** cloudera**</id** **>**
18. **<url>** https://repository.cloudera.com/artifactory/cloudera-repos/. **</url>**
19. **</repository>**
20. **<repository>**
21. **<id>** jboss**</id** **>**
22. **<url>** http://repository.jboss.org/nexus/content/groups/public-jboss **</url>**
23. **</repository>**
24. **<repository>**
25. **<id>** Sonatype snapshots **</id>**
26. **<url>** http://oss.sonatype.org/content/repositories/snapshots/**</url** **>**
27. **</repository>**
28. **</repositories** **>**
30. **<build** **>**
31. **<sourceDirectory>** src/spark/**</sourceDirectory** **>**
32. **<testSourceDirectory>** src/test/**</testSourceDirectory** **>**
33. <!-- 添加pluginManagement：解决 Maven报Plugin execution not covered by lifecycle configuration
34. 具体详见： http://blog.csdn.net/xxd851116/article/details/25197373
35. --**>**
36. **<pluginManagement>**
37. **<plugins>**
38. **<plugin>**
39. **<groupId>** org.scala-tools**</groupId** **>**
40. **<artifactId>** maven- scala-plugin**</artifactId** **>**
41. **<executions>**
42. **<execution>**
43. **<goals>**
44. **<goal>** compile**</** **goal>**
45. **<goal>** testCompile**</goal** **>**
46. **</goals>**
47. **</execution>**
48. **</executions>**
49. **<configuration>**
50. **<scalaVersion>** 2.10.5**</** **scalaVersion>**
51. **</configuration>**
52. **</plugin>**
54. **<plugin>**
55. **<groupId>** org.apache.maven.plugins**</groupId** **>**
56. **<artifactId>** maven-shade- plugin**</artifactId>**
57. **<version>** 2.2**</** **version>**
58. **<executions>**
59. **<execution>**
60. **<phase>** package**</** **phase>**
61. **<goals>**
62. **<goal>** shade**</** **goal>**
63. **</goals>**
64. **<configuration>**
65. **<filters>**
66. **<filter>**
67. **<artifact>** \*:\***</** **artifact>**
68. **<excludes>**
69. **<exclude>** \***</** **exclude>**
70. **</excludes>**
71. **</filter>**
72. **</filters>**
73. **<transformers>**
75. **<transformer**
76. implementation="org.apache.maven.plugins.shade.resource.AppendingTransformer" **>**
77. **<resource>** reference.conf**</resource** **>**
78. **</transformer>**
80. **<transformer**
81. implementation="org.apache.maven.plugins.shade.resource.ManifestResourceTransformer" **>**
82. **</transformer>**
83. **</transformers>**
84. **</configuration>**
85. **</execution>**
86. **</executions>**
87. **</plugin>**
88. **</plugins>**
89. **</pluginManagement>**
90. **</build** **>**
92. **<dependencies** **>**
93. <!-- 解决 maven引用jdk中的tools.jar报Missing artifact的问题 ,指定maven去本地寻找 tools.jar -->
94. **<dependency>**
95. **<groupId>** jdk.tools**</groupId** **>**
96. **<artifactId>** jdk.tools**</artifactId** **>**
97. **<version>** 1.7**</** **version>**
98. **<scope>** system**</** **scope>**
99. **<systemPath>** ${JAVA\_HOME}/lib/tools.jar **</systemPath>**
100. **</dependency>**
101. **<dependency>**
102. **<groupId>** org.apache.spark**</groupId** **>**
103. **<artifactId>** spark-core\_2.10**</artifactId** **>**
104. **<version>** 1.2.0-cdh5.3.2**</version** **>**
105. **</dependency>**
106. **<dependency>**
107. **<groupId>** org.apache.hadoop**</groupId** **>**
108. **<artifactId>** hadoop-client **</artifactId>**
109. **<version>** 2.6.0-cdh5.3.6**</version** **>**
110. **</dependency>**
111. **<dependency>**
112. **<groupId>** org.apache.spark**</groupId** **>**
113. **<artifactId>** spark-streaming\_2.10**</artifactId** **>**
114. **<version>** 0.9.0-cdh5.0.0**</version** **>**
115. **</dependency>**
116. **<dependency>**
117. **<groupId>** org.apache.spark**</groupId** **>**
118. **<artifactId>** spark-yarn\_2.10**</artifactId** **>**
119. **<version>** 1.2.0-cdh5.3.3**</version** **>**
120. **</dependency>**
121. **<dependency>**
122. **<groupId>** org.apache.spark**</groupId** **>**
123. **<artifactId>** spark-tools\_2.10**</artifactId** **>**
124. **<version>** 0.9.0-cdh5.0.0**</version** **>**
125. **</dependency>**
127. **</dependencies** **>**
129. **</project>**

注意：

（1）使用CDH集成的相关包，其中版本号位2.10 版本，故这里的scala使用2.10.x系列，若果使用高的版本，会出现无法编译。

（2）当网络很慢比较的情况下，会出现很多的错误，无论怎么样，请耐心等待

（3）org.apache.spark 下的包我们使用的是2.10，这里我们使用scala版本2.10 相对应。

（4）<pluginManagement> 解决插件报错问题

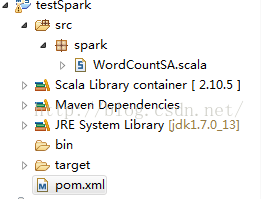
2.2 开发Spark的WordCount程序

Eclipse版本：4.4.2

scala版本：2.10.5。这里尝试使用过2.11.x，出现cdh集成的spar相关的包是无发编译

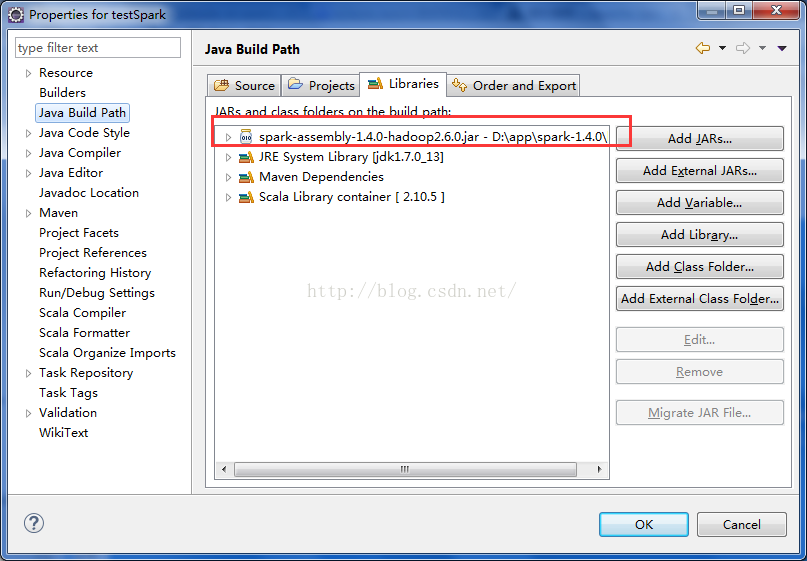
JDK版本：1.7.0

新建项目步骤： 新建scala project->拷贝pom.xml 到scala project中->转成maven项目



2.3 导出jar包

（1） 引入spark-assembly-1.4.0-hadoop2.6.0.jar



（2）在Scala工程中，右击“WordCountSA”，选择“Export”，并在弹出框中选择“Java” –> “JAR File”，进而将该程序编译成jar包，可以起名为“spark-wordcount-in-scala.jar”，我导出的jar包下载地址是 [spark-wordcount-in-scala.jar](http://kanboxshare.com/link/khGIHySIDoEg8SMP96C2PUVeLmQrHyI9BeZFGMOmRX9svM3UOu2dPT66Md4a0ZkQCKZeGiKJvHfUvM3wbLO4XeorAn3V2kvZ8V7Qq)

参考： http://dongxicheng.org/framework-on-yarn/spark-eclipse-ide/

3、spark standalone 模式执行wordCount

3.1 启动hdfs

start-dfs.sh

3.2 启动spark

cd $SPARK\_HOME/sbin

start-all.sh

cd $SPARK\_HOME/bin

spark-shell

3.3 执行wordCount

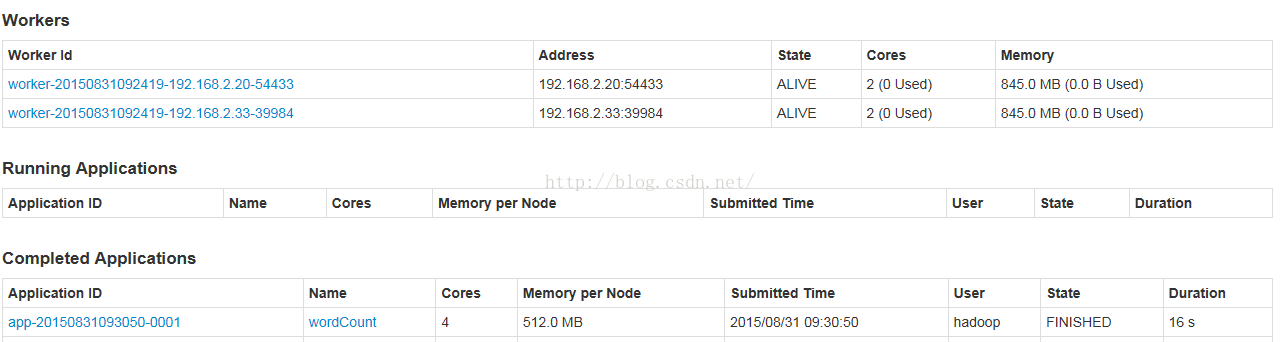
[hadoop@mycluster spark]$ ./bin/spark-submit --class spark.WordCountSA /home/hadoop/testSpark-1.0-SNAPSHOT.jar hdfs://mycluster:9000/wc.txt hdfs://mycluster:9000/output2

3.4 验证结果

[hadoop@mycluster ~]$ hdfs dfs -cat /output2/part\*  
(hello,4)  
(you,2)  
(china,1)  
(me,1)

3.5 登录spark ui 任务页面

http://192.168.2.20:8080/



4、spark在提交任务时，出现如下错误

WARN TaskSchedulerImpl: Initial job has not accepted any resources; check your cluster UI to ensure that workers are registered and have sufficient memory 15/03/26 22:29:51 WARN TaskSchedulerImpl: Initial job has not accepted any resources; check your cluster UI to ensure that workers are registered and have sufficient memory 15/03/26 22:30:06 WARN TaskSchedulerImpl: Initial job has not accepted any resources; check your cluster UI to ensure that workers are registered and have sufficient memory 15/03/26 22:30:21 WARN TaskSchedulerImpl: Initial job has not accepted any resources; check your cluster UI to ensure that workers are registered and have sufficient memory

从警告信息上看，初始化job时没有获取到任何资源；提示检查集群，确保workers可以被注册并有足够的内存资源。

如上问题产生的原因是多方面的，可能原因如下：

1.因为提交任务的节点不能和spark工作节点交互，因为提交完任务后提交任务节点上会起一个进程，展示任务进度，大多端口为4044，工作节点需要反馈进度给该该端口，所以如果主机名或者IP在hosts中配置不正确。所以检查下主机名和ip是否配置正确

2.有可能是内存不足

  检查内存

  conf.set("spark.executor.memory", "3000m")

  Make sure to set SPARK\_LOCAL\_IP andSPARK\_MASTER\_IP.

  查看8080端口，确保一些workers保持Alive状态，确保 some cores 是可利用的。

[**012-01Spark On YARN 环境搭建**](http://blog.csdn.net/shenfuli/article/details/48162949)

1、Scala 安装

http://www.scala-lang.org/files/archive/scala-2.10.4.tgz

tar -zxvf scala-2.10.4.tgz -C app/  
cd  app  
ln -s scala-2.10.4 scala

2、Spark 安装

**tar -zxvf**spark-1.4.0-bin-hadoop2.6.tgz -C app

ln -s spark-1.4.0-bin-hadoop2.6 spark

# vim spark-env.sh  
export JAVA\_HOME=/home/hadoop/app/jdk1.7.0\_76  
export SCALA\_HOME=/home/hadoop/app/scala  
export HADOOP\_HOME=/home/hadoop/app/hadoop-2.6.0

## worker节点的主机名列表  
# vim slaves

192.168.2.20  
192.168.2.33

# mv log4j.properties.template log4j.properties

## 在Master节点上执行

 cd  $SPARK\_HOME/bin

./start-all.sh

**3、配置系统环境变量**

vim /etc/profile  
export SCALA\_HOME=/home/hadoop/app/scala  
export SPARK\_HOME=/home/hadoop/app/spark  
export PATH=$PATH:$HIVE\_HOME/bin:$HBASE\_HOME/bin:$SCALA\_HOME/bin:$SPARK\_HOME/bin:$SPARK\_HOME/sbin

source /etc/profile

4、相关测试## 监控页面URL  
http://192.168.2.20:8080/

## 先切换到“cd $SPARK\_HOME”目录

**(1)、本地模式**  
#进行spark-shell命令

./spark-shell

#测试

sc.textFile("/home/hadoop/wc.txt").flatMap( line=>line.split("\t") ).map( word=>(word,1) ).reduceByKey(\_ + \_).collect

#验证

<http://192.168.2.20:4040/>

**(2)、** 基于YARN模式

cd $SPARK\_HOME  
bin/spark-submit  --class  org.apache.spark.examples.SparkPi \  
--master yarn-cluster \  
--num-executors 3 \  
--driver-memory 1g \  
--executor-memory 1g \  
--executor-cores 1 \  
lib/spark-examples\*.jar  10

执行步骤出现的日志

[hadoop@mycluster spark]$ bin/spark-submit  --class  org.apache.spark.examples.SparkPi \  
> --master yarn-cluster \  
> --num-executors 3 \  
> --driver-memory 1g \  
> --executor-memory 1g \  
> -executor-cores 1 \  
> lib/spark-examples\*.jar  10  
Error: Unrecognized option '-executor-cores'.  
Run with --help for usage help or --verbose for debug output  
[hadoop@mycluster spark]$ bin/spark-submit  --class  org.apache.spark.examples.SparkPi \  
> --master yarn-cluster \  
> --num-executors 3 \  
> --driver-memory 1g \  
> --executor-memory 1g \  
> --executor-cores 1 \  
> lib/spark-examples\*.jar  10  
15/08/30 22:53:29 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable  
15/08/30 22:53:29 INFO RMProxy: Connecting to ResourceManager at mycluster/192.168.2.20:8032  
15/08/30 22:53:29 INFO Client: Requesting a new application from cluster with 1 NodeManagers  
15/08/30 22:53:29 INFO Client: Verifying our application has not requested more than the maximum memory capability of the cluster (8192 MB per container)  
15/08/30 22:53:29 INFO Client: Will allocate AM container, with 1408 MB memory including 384 MB overhead  
15/08/30 22:53:29 INFO Client: Setting up container launch context for our AM  
15/08/30 22:53:29 INFO Client: Preparing resources for our AM container  
15/08/30 22:53:30 INFO Client: Uploading resource file:/home/hadoop/app/spark-1.4.0-bin-hadoop2.6/lib/spark-assembly-1.4.0-hadoop2.6.0.jar -> hdfs://mycluster:9000/user/hadoop/.sparkStaging/application\_1440995865051\_0005/spark-assembly-1.4.0-hadoop2.6.0.jar  
15/08/30 22:53:33 INFO Client: Uploading resource file:/home/hadoop/app/spark-1.4.0-bin-hadoop2.6/lib/spark-examples-1.4.0-hadoop2.6.0.jar -> hdfs://mycluster:9000/user/hadoop/.sparkStaging/application\_1440995865051\_0005/spark-examples-1.4.0-hadoop2.6.0.jar  
15/08/30 22:53:39 INFO Client: Uploading resource file:/tmp/spark-ecb5f2dc-f66b-42e6-a8ae-befce75074c0/\_\_hadoop\_conf\_\_846873578807129658.zip -> hdfs://mycluster:9000/user/hadoop/.sparkStaging/application\_1440995865051\_0005/\_\_hadoop\_conf\_\_846873578807129658.zip  
15/08/30 22:53:40 INFO Client: Setting up the launch environment for our AM container  
15/08/30 22:53:40 INFO SecurityManager: Changing view acls to: hadoop  
15/08/30 22:53:40 INFO SecurityManager: Changing modify acls to: hadoop  
15/08/30 22:53:40 INFO SecurityManager: SecurityManager: authentication disabled; ui acls disabled; users with view permissions: Set(hadoop); users with modify permissions: Set(hadoop)  
15/08/30 22:53:40 INFO Client: Submitting application 5 to ResourceManager  
15/08/30 22:53:40 INFO YarnClientImpl: Submitted application application\_1440995865051\_0005  
15/08/30 22:53:41 INFO Client: Application report for application\_1440995865051\_0005 (state: ACCEPTED)  
15/08/30 22:53:41 INFO Client:  
         client token: N/A  
         diagnostics: N/A  
         ApplicationMaster host: N/A  
         ApplicationMaster RPC port: -1  
         queue: default  
         start time: 1441000420286  
         final status: UNDEFINED  
         tracking URL: http://mycluster:8088/proxy/application\_1440995865051\_0005/  
         user: hadoop  
15/08/30 22:53:43 INFO Client: Application report for application\_1440995865051\_0005 (state: ACCEPTED)  
15/08/30 22:53:45 INFO Client: Application report for application\_1440995865051\_0005 (state: ACCEPTED)  
15/08/30 22:53:46 INFO Client: Application report for application\_1440995865051\_0005 (state: ACCEPTED)  
15/08/30 22:53:48 INFO Client: Application report for application\_1440995865051\_0005 (state: ACCEPTED)  
15/08/30 22:53:50 INFO Client: Application report for application\_1440995865051\_0005 (state: ACCEPTED)  
15/08/30 22:53:52 INFO Client: Application report for application\_1440995865051\_0005 (state: ACCEPTED)  
15/08/30 22:53:54 INFO Client: Application report for application\_1440995865051\_0005 (state: ACCEPTED)  
15/08/30 22:53:56 INFO Client: Application report for application\_1440995865051\_0005 (state: ACCEPTED)  
15/08/30 22:53:57 INFO Client: Application report for application\_1440995865051\_0005 (state: ACCEPTED)  
15/08/30 22:53:58 INFO Client: Application report for application\_1440995865051\_0005 (state: ACCEPTED)  
15/08/30 22:53:59 INFO Client: Application report for application\_1440995865051\_0005 (state: ACCEPTED)  
15/08/30 22:54:00 INFO Client: Application report for application\_1440995865051\_0005 (state: ACCEPTED)  
15/08/30 22:54:01 INFO Client: Application report for application\_1440995865051\_0005 (state: ACCEPTED)  
15/08/30 22:54:02 INFO Client: Application report for application\_1440995865051\_0005 (state: ACCEPTED)  
15/08/30 22:54:03 INFO Client: Application report for application\_1440995865051\_0005 (state: ACCEPTED)  
15/08/30 22:54:04 INFO Client: Application report for application\_1440995865051\_0005 (state: RUNNING)  
15/08/30 22:54:04 INFO Client:  
         client token: N/A  
         diagnostics: N/A  
         ApplicationMaster host: 192.168.2.20  
         ApplicationMaster RPC port: 0  
         queue: default  
         start time: 1441000420286  
         final status: UNDEFINED  
         tracking URL: http://mycluster:8088/proxy/application\_1440995865051\_0005/  
         user: hadoop  
15/08/30 22:54:05 INFO Client: Application report for application\_1440995865051\_0005 (state: RUNNING)  
15/08/30 22:54:06 INFO Client: Application report for application\_1440995865051\_0005 (state: RUNNING)  
15/08/30 22:54:07 INFO Client: Application report for application\_1440995865051\_0005 (state: RUNNING)  
15/08/30 22:54:08 INFO Client: Application report for application\_1440995865051\_0005 (state: RUNNING)  
15/08/30 22:54:09 INFO Client: Application report for application\_1440995865051\_0005 (state: RUNNING)  
15/08/30 22:54:10 INFO Client: Application report for application\_1440995865051\_0005 (state: RUNNING)  
15/08/30 22:54:11 INFO Client: Application report for application\_1440995865051\_0005 (state: RUNNING)  
15/08/30 22:54:12 INFO Client: Application report for application\_1440995865051\_0005 (state: RUNNING)  
15/08/30 22:54:13 INFO Client: Application report for application\_1440995865051\_0005 (state: RUNNING)  
15/08/30 22:54:15 INFO Client: Application report for application\_1440995865051\_0005 (state: RUNNING)  
15/08/30 22:54:17 INFO Client: Application report for application\_1440995865051\_0005 (state: RUNNING)  
15/08/30 22:54:18 INFO Client: Application report for application\_1440995865051\_0005 (state: RUNNING)  
15/08/30 22:54:19 INFO Client: Application report for application\_1440995865051\_0005 (state: RUNNING)  
15/08/30 22:54:20 INFO Client: Application report for application\_1440995865051\_0005 (state: RUNNING)  
15/08/30 22:54:21 INFO Client: Application report for application\_1440995865051\_0005 (state: RUNNING)  
15/08/30 22:54:23 INFO Client: Application report for application\_1440995865051\_0005 (state: RUNNING)  
15/08/30 22:54:24 INFO Client: Application report for application\_1440995865051\_0005 (state: RUNNING)  
15/08/30 22:54:25 INFO Client: Application report for application\_1440995865051\_0005 (state: RUNNING)  
15/08/30 22:54:26 INFO Client: Application report for application\_1440995865051\_0005 (state: RUNNING)  
15/08/30 22:54:27 INFO Client: Application report for application\_1440995865051\_0005 (state: RUNNING)  
15/08/30 22:54:29 INFO Client: Application report for application\_1440995865051\_0005 (state: RUNNING)  
15/08/30 22:54:30 INFO Client: Application report for application\_1440995865051\_0005 (state: RUNNING)  
15/08/30 22:54:31 INFO Client: Application report for application\_1440995865051\_0005 (state: RUNNING)  
15/08/30 22:54:33 INFO Client: Application report for application\_1440995865051\_0005 (state: RUNNING)  
15/08/30 22:54:34 INFO Client: Application report for application\_1440995865051\_0005 (state: RUNNING)  
15/08/30 22:54:36 INFO Client: Application report for application\_1440995865051\_0005 (state: RUNNING)  
15/08/30 22:54:37 INFO Client: Application report for application\_1440995865051\_0005 (state: RUNNING)  
15/08/30 22:54:38 INFO Client: Application report for application\_1440995865051\_0005 (state: RUNNING)  
15/08/30 22:54:40 INFO Client: Application report for application\_1440995865051\_0005 (state: RUNNING)  
15/08/30 22:54:41 INFO Client: Application report for application\_1440995865051\_0005 (state: RUNNING)  
15/08/30 22:54:42 INFO Client: Application report for application\_1440995865051\_0005 (state: RUNNING)  
15/08/30 22:54:43 INFO Client: Application report for application\_1440995865051\_0005 (state: FINISHED)  
15/08/30 22:54:43 INFO Client:  
         client token: N/A  
         diagnostics: N/A  
         ApplicationMaster host: 192.168.2.20  
         ApplicationMaster RPC port: 0  
         queue: default  
         start time: 1441000420286  
         final status: SUCCEEDED  
         tracking URL: http://mycluster:8088/proxy/application\_1440995865051\_0005/A  
         user: hadoop  
15/08/30 22:54:43 INFO Utils: Shutdown hook called  
15/08/30 22:54:43 INFO Utils: Deleting directory /tmp/spark-ecb5f2dc-f66b-42e6-a8ae-befce75074c0

常见问题：

基于YARN模式下执行上述spark-submit，出现下面的错误

[hadoop@mycluster spark]$ ./bin/spark-submit --class org.apache.spark.examples.SparkPi   --master yarn-cluster   --master yarn-cluster 10  
Exception in thread "main" java.lang.Exception: When running with master 'yarn-cluster' either HADOOP\_CONF\_DIR or YARN\_CONF\_DIR must be set in the environment.  
        at org.apache.spark.deploy.SparkSubmitArguments.validateSubmitArguments(SparkSubmitArguments.scala:239)  
        at org.apache.spark.deploy.SparkSubmitArguments.validateArguments(SparkSubmitArguments.scala:216)  
        at org.apache.spark.deploy.SparkSubmitArguments.<init>(SparkSubmitArguments.scala:103)  
        at org.apache.spark.deploy.SparkSubmit$.main(SparkSubmit.scala:106)  
        at org.apache.spark.deploy.SparkSubmit.main(SparkSubmit.scala)  
15/08/30 22:25:45 INFO Utils: Shutdown hook called

解决方案： 配置 HADOOP\_CONF\_DIR or YARN\_CONF\_DIR 和变量，如下

cd $SPARK\_HOME/conf

vi spark-env.sh   
# Options read in YARN client mode  
# - HADOOP\_CONF\_DIR, to point Spark towards Hadoop configuration files  
HADOOP\_CONF\_DIR=/home/hadoop/app/hadoop-2.6.0/etc/hadoop

[**012-02Spark On YARN Scala 开发WordCount计数**](http://blog.csdn.net/shenfuli/article/details/48162963)

1、使用spark-submit方式提交代码

注：由于通过Spark on yarn，使用该命令时候需要指定 --master

2、编写提交Yarn的wordcount程序

其中：standalon代码的区别： 没有conf配置信息，所有的配置和资源通过yarn来完成

package spark  
  
import org.apache.spark.SparkContext  
import org.apache.spark.SparkContext.\_  
/\*\*  
\* Spark on yarn 模式单词计数,并倒序排序  
\* @author shenfl  
\*/  
object WordCountSparkOnYarn {  
   
  def main(args: Array[String]): Unit = {  
    if(args.length!=2){  
      println("Usage: spark-submit --class spark.WordCountSparkOnYarn wordCount.jar inputpath outputpath ")  
      System.exit(0);  
    }  
     
    val sc = new SparkContext;  
    val files = sc.textFile(args(0));  
     
    files.flatMap( (\_.split("\t")) ).map( (\_,1) ).reduceByKey(\_ + \_)  
         .map( x => (x.\_2,x.\_1) ).sortByKey(false)  
         .map( x =>(x.\_2,x.\_1))  
         .repartition(1)  
         .saveAsTextFile(args(1))  
  }  
}

3、Spark on yarn 模式下运行

3.1 编译成jar包 testSpark-1.0-SNAPSHOT.jar

3.2 上传服务器并通过spark submit在yarn上运行

spark-submit --class spark.WordCountSparkOnYarn --master yarn-cluster  testSpark-1.0-SNAPSHOT.jar  wc.txt output3

注： wc.txt 的数据为：

通过执行hdfs dfs -cat wc.txt命令，查看hdfs上数据为：  
hello   you  
hello   m

3.3 检查结果

hdfs dfs -cat output4/part\*  
(hello,2)  
(you,1)  
(me,1)

注意： 程序引入reduceByKey(\_ + \_) 出错是因为没有导入对应的包，例如：  
import org.apache.spark.SparkContext  
import org.apache.spark.SparkContext.\_

# [015-Spark SQL与 Hive集成](http://blog.csdn.net/shenfuli/article/details/48163027)

1.spark和hive版本要求  
    spark1.4.0  
    hive-0.13.0（注意：不要使用hive0.14版本用于集成，否则会出很多问题）  
  
  
2、集群规划(hadoop 采用伪分布式）  
  
2.1 hadoop 伪分布式安装规划（同hadoop集群模式一样）  
NameNode: 192.168.2.20  
DataNode:192.168.2.20  
ResourceManager:192.168.2.20  
NodeManager:192.168.2.20  
  
2.2 Spark 集群规划  
Master: 192.168.2.20   192.168.2.33  
Worker:192.168.2.33  
  
  
2.3 Hive规划  
hive在192.168.2.20 和 192.168.2.33 两台节点上都要安装  
  
备注： hive使用mysql为存储元数据

**[html]** [view plain](http://blog.csdn.net/shenfuli/article/details/48163027) [copy](http://blog.csdn.net/shenfuli/article/details/48163027)

[print?](http://blog.csdn.net/shenfuli/article/details/48163027)

1. [hadoop@mycluster conf]$ more hive-site.xml
2. **<configuration>**
3. **<property>**
4. **<name>**javax.jdo.option.ConnectionURL**</name>**
5. **<value>**jdbc:mysql://mycluster:3306/hive?createDatabaseIfNotExist=true**</value>**
6. **</property>**
8. **<property>**
9. **<name>**javax.jdo.option.ConnectionDriverName**</name>**
10. **<value>**com.mysql.jdbc.Driver**</value>**
11. **</property>**
13. **<property>**
14. **<name>**javax.jdo.option.ConnectionUserName**</name>**
15. **<value>**root**</value>**
16. **</property>**
18. **<property>**
19. **<name>**javax.jdo.option.ConnectionPassword**</name>**
20. **<value>**123**</value>**
21. **</property>**
22. **</configuration>**

3、Spark 参数配置

3.1 在$SPARK\_HOME/conf下，修改spark-env.sh

**[html]** [view plain](http://blog.csdn.net/shenfuli/article/details/48163027) [copy](http://blog.csdn.net/shenfuli/article/details/48163027)

[print?](http://blog.csdn.net/shenfuli/article/details/48163027)

1. [hadoop@mycluster ~]$ cd $SPARK\_HOME
2. [hadoop@mycluster spark]$ more conf/spark-env.sh
3. #!/usr/bin/env bash
4. SPARK\_MASTER\_IP=mycluster
5. export JAVA\_HOME=/home/hadoop/app/jdk1.7.0\_76
6. export SCALA\_HOME=/home/hadoop/app/scala
7. export HADOOP\_HOME=/home/hadoop/app/hadoop-2.6.0
8. export HIVE\_HOME=/home/hadoop/app/hive
9. export SPARK\_CLASSPATH=$HIVE\_HOME/lib/mysql-connector-java-5.1.28.jar

注意： 所有spark节点都要相同的配置

3.2 把hive的hive-site.xml拷贝到$SPARK\_HOME/conf下

3.3 .把mysql的驱动包拷贝到$SPARK\_HOME/lib下  
  
3.4 spark会打印许多INFO级别的日志，可以修改log4j.prop文件中的日志级别 设置warn

[hadoop@mycluster conf]$ more log4j.properties  
# Set everything to be logged to the console  
log4j.rootCategory=WARN , console  
log4j.appender.console=org.apache.log4j.ConsoleAppender  
log4j.appender.console.target=System.err  
log4j.appender.console.layout=org.apache.log4j.PatternLayout  
log4j.appender.console.layout.ConversionPattern=%d{yy/MM/dd HH:mm:ss} %p %c{1}: %m%n  
  
# Settings to quiet third party logs that are too verbose  
log4j.logger.org.spark-project.jetty=WARN  
log4j.logger.org.spark-project.jetty.util.component.AbstractLifeCycle=ERROR  
log4j.logger.org.apache.spark.repl.SparkIMain$exprTyper=INFO  
log4j.logger.org.apache.spark.repl.SparkILoop$SparkILoopInterpreter=INFO

4、启动 spark-sql

4.1 启动spark集群

cd $SPARK\_HOME

sbin/start-all.sh

4.2 启动spark-sql

bin/spark-sql

5、验证spark-sql

5.1 登录hive进行查询

（1）登录hive命令

[hadoop@mycluster hive]$ bin/hive

（2）使用mydb数据库和显示所有表

hive (default)> use mydb;  
OK  
Time taken: 1.833 seconds  
hive (mydb)> show tables;  
OK  
tab\_name  
access\_log  
bucket\_table  
jfapp\_pv  
order  
stu  
stu\_index\_table  
stu\_view  
t1  
Time taken: 0.256 seconds, Fetched: 8 row(s)

（3）使用count函数

hive (mydb)> select count(\*) from stu;  
Total jobs = 1  
Launching Job 1 out of 1  
Number of reduce tasks determined at compile time: 1  
In order to change the average load for a reducer (in bytes):  
  set hive.exec.reducers.bytes.per.reducer=<number>  
In order to limit the maximum number of reducers:  
  set hive.exec.reducers.max=<number>  
In order to set a constant number of reducers:  
  set mapreduce.job.reduces=<number>  
Starting Job = job\_1441094211714\_0002, Tracking URL = http://mycluster:8088/proxy/application\_1441094211714\_0002/  
Kill Command = /home/hadoop/app/hadoop-2.6.0/bin/hadoop job  -kill job\_1441094211714\_0002  
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1  
2015-09-01 04:05:58,315 Stage-1 map = 0%,  reduce = 0%  
2015-09-01 04:06:18,500 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 2.54 sec  
2015-09-01 04:06:28,524 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 4.78 sec  
MapReduce Total cumulative CPU time: 4 seconds 780 msec  
Ended Job = job\_1441094211714\_0002  
MapReduce Jobs Launched:  
Job 0: Map: 1  Reduce: 1   Cumulative CPU: 4.78 sec   HDFS Read: 291 HDFS Write: 2 SUCCESS  
Total MapReduce CPU Time Spent: 4 seconds 780 msec  
OK  
\_c0  
9  
Time taken: 85.028 seconds, Fetched: 1 row(s)

注：hive的count需要通过mapreduce完成，效率上很低。

5.2 登录spark-sql如同hive一样操作

（1）登录hive命令

[hadoop@mycluster spark]$ bin/spark-sql

（2）使用mydb数据库和显示所有表

spark-sql (default)> use mydb;  
OK  
result  
Time taken: 1.293 seconds  
spark-sql (default)> show tables;  
tableName       isTemporary  
access\_log      false  
bucket\_table    false  
jfapp\_pv        false  
order   false  
stu     false  
stu\_index\_table false  
stu\_view        false  
t1      false  
Time taken: 0.195 seconds, Fetched 8 row(s)

（3） 使用count函数

spark-sql (default)> select count(\*) from stu;  
\_c0  
9  
Time taken: 5.894 seconds, Fetched 1 row(s)

通过Spark-sql和hive计算的结果对比，使用spark-sql性能比hive要高很多，将近85:5约为16倍