1. Spark 调研

北京网感至察科技有限公司

2015年4月1日

* 1. 运行环境安装
     + 1. 操作系统

此系统要求运行于基于Unix的操作系统（Linux, Max Os等)之上，测试环境为Ubuntu 12.04 LTS.

* + - 1. 必备软件库与语言环境

Spark的运行环境要求JDK 1.6+, Scala 2.9+. 在缺省安装上述操作系统之后，运行

java -version

scala -version

检查JDK和Scala环境。若不符合要求，以下提供了JDK和Scala环境的安装步骤，对应的软件库为jdk1.7.0\_60和scala-2.10.5. sbt提供了类似Maven的项目构建功能，推荐安装。测试的Spark版本为对应Hadoop2.4的预编译版本Spark1.3.0-bin-hadoop2.4.

1. [可选] JDK 安装:

前往Oracle网站下载jdk安装包，假定为jdk1.7.0\_60.tar.gz, 下载路径为~/Downloads

运行：

cd /usr/lib/jvm/

sudo mv ~/Downloads/JDK.tar.gz ./ (需要提供管理员密码）

tar zxvf jdk1.7.0\_60.tar.gz

2. [可选] Scala安装：

运行：

cd /usr/local/lib

wget http://www.scala-lang.org/files/archive/scala-2.10.5.tgz

tar zxvf scala-2.10.5.tgz

3. Spark安装：

cd /usr/local/lib

wget http://www.apache.org/dyn/closer.cgi/spark/spark-1.3.0/spark-1.3.0-bin-hadoop2.4.tgz

tar zxvf spark-1.3.0-bin-hadoop2.4.tgz

4. [可选] sbt安装：

echo "deb https://dl.bintray.com/sbt/debian /" | sudo tee -a /etc/apt/sources.list.d/sbt.list

sudo apt-get update

sudo apt-get install sbt

5. 系统环境变量配置:

编辑~/.bashrc文件，在文件末尾添加：

export JAVA\_HOME="/usr/lib/jvm/jdk1.7.0\_60"

export SCALA\_HOME="/usr/local/lib/scala-2.10.5"

export SPARK\_HOME="/usr/local/lib/spark-1.3.0-bin-hadoop2.4/"

export PATH="$JAVA\_HOME/bin:$SPARK\_HOME/bin:$SCALA\_HOME/bin:$PATH"

* 1. Spark配置与测试运行（单机）

以下以WordCount为例，测试Spark的交互运行和创建应用两种运行方式。

* + - 1. REPL交互运行

进入Spark Shell：

spark-shell

Spark中，RDD的创建有两种方式，分别是从输入流读取（比如HDFS file）和从其他RDD转化。假定待处理文件为/home/sid/test.txt, 文件内每行为一个单词。

在Spark Shell内运行：

val lines = sc.textFile("/home/sid/test.txt")

val count = lines.flatMap(line => line.split(" " )).map(word => (word, 1)).reduceByKey(\_+\_)

这里， flatMap, map, reduceByKey操作都生成了一个新的RDD，最终生成count，其形式为(String, Int)元组，对应每个单词出现的次数：

count.foreach(println(\_))

可以将每个元组依次输出。

* + - 1. 创建Spark应用

Spark应用通过spark-submit进行提交。编写scala程序时，需要导入 SparkContext和 SparkConf.

SimpleApp.scala程序：

import org.apache.spark.SparkContext

import org.apache.spark.SparkContext.\_

import org.apache.spark.SparkConf

object SimpleApp {

def main(args: Array[String]) {

val filename = "/home/sid/test.txt"

val conf = new SparkConf().setAppName("Word Count")

val sc = new SparkContext(conf)

val lines = sc.textFile(filename).cache()

val counts = lines.flatMap(line => line.split(" ")).map(word => (word, 1)).reduceByKey(\_+\_)

counts.foreach(println(\_))

}

}

Simple.sbt配置文件：

name := "Simple Project"

version := "1.0"

scalaVersion := "2.10.5"

libraryDependencies += "org.apache.spark" %% "spark-core" % "1.3.0"

运行sbt编译：

sbt package

编译结束后，生成jar文件，假定为simple-project\_2.10-1.0.jar.

利用spark-submit提交，提交时可以制定参数：

spark-submit –master local[4] simple-project\_2.10-1.0.jar

* 1. Postgre SQL数据导入

首先安装PostgreSQL, 运行：

sudo apt-get install postgresql-9.1

假定数据库用户名postgres, 密码为123, 待导入的数据sql文件为gflux.sql，运行：

psql -U postgres -h localhost -d gflux -f gflux.sql

则sql文件里的数据可以导入至PostgreSQL数据库

* 1. Spark中生成DataFrame

DataFrame是Spark中对关系型数据结构的抽象，可以类比为数据库中的一个表。Spark1.3支持创建DataFrame的方式有以下4种：结构化的数据文件(比如JSON); Hive表; 外部数据库(通过JDBC); 已有的RDD.

1. 下载对应的jdbc驱动

假定下载文件为postgresql-9.1-903.jdbc4.jar，文件存储至$SPARK\_HOME/lib文件夹下。

2. 指定Spark的classpath：

编辑$SPARK\_HOME/conf下spark\_env.sh文件，加入：

export SPARK\_CLASSPATH=.:$SPARK\_HOME/lib/postgresql-9.1-903.jdbc4.jar:$SPARK\_CLASSPATH

3. 生成DataFrame (以java为例，读取导入数据中里的item表)

**import** java.util.HashMap;

**import** java.util.Map;

**import** org.apache.spark.api.java.\*;

**import** org.apache.spark.SparkConf;

**import** org.apache.spark.sql.DataFrame;

**import** org.apache.spark.sql.SQLContext;

**public** **class** Test {

**public** **static** **void** **main**(String[] **args**) {

//创建SparkContext上下文

JavaSparkContext **sc** = **new** JavaSparkContext(**new** SparkConf().setAppName("JDBC demo"));

//创建SQLContext上下文

SQLContext **sqc** = **new** SQLContext(sc);

//创建选项字典

Map<String, String> **options** = **new** HashMap<String, String>();

String **user** = "postgres";

String **password** = "123";

options.put("url", "jdbc:postgresql://localhost:5432/gflux?user="

+ user + "&password=" + password);

options.put("dbtable", "item");

//生成DataFrame

DataFrame **jdbcDF** = sqc.load("jdbc", options);

//展示DataFrame

jdbcDF.show(); //show()方法只显示20项

}

}

最后运行结果应为：



* 1. 简单统计运算

随机变量相关性

我们一般用Pearson相关系数（以下简称为相关系数）来衡量两个随机变量的相关性。假设两个随机变量分别为x，y，则它们的相关系数的定义为：



其中，cov(X, Y)代表x，y的协方差，\sigma\_x 和 \sigma\_y 分别代表x，y的标准差，其定义为



其中, \mu 代表随机变量X的期望， N代表样本数量。

严格地说，相关系数的大小并不能直接衡量两个随机变量的相关性，而是需要根据场景决定；但是，一般经验而言，当相关系数低于0.4我们认为两变量相关性不大，而相关系数超过0.7时我们认为两变量有较大的相关性。

在程序中，通过简单的数学运算可以求得两列序列的相关性，事实上，Spark已经提供了相关功能，其Scala API如下：

val correlation: Double = Statistics.corr(seriesX, seriesY, "pearson")

//第三个参数perason可以省略; seriesX和seriesY的类型是RDD[Double]

以下例子演示了从数据库里读取数据，取出两列，转换为RDD[Double]形式后求相关系数的过程。计算的两个变量分别为支付数额和购买数量。

**import** org.apache.spark.SparkContext

**import** org.apache.spark.SparkConf

**import** org.apache.spark.sql.SQLContext

**import** org.apache.spark.mllib.stat.Statistics

**import** org.apache.spark.sql.Row

**object** TestCorr {

**def** main(args: Array[String]) {

// 创建上下文

**val** conf = **new** SparkConf().setAppName("Correlation")

conf.setMaster("local")

**val** sc = **new** SparkContext(conf)

**val** sqc = **new** SQLContext(sc)

// 从数据库里获取DataFrame

**val** df = sqc.load("jdbc",

Map("url" -> "jdbc:postgresql://localhost:5432/gflux?user=postgres&password=123",

"dbtable" -> "fact\_trans"))

// 分别选取两列，并通过RDD的map()方法转换将RDD[Row]转换为RDD[Double]

**val** pay = df.select("pay").rdd.cache().map{ row => row.getDouble(0) }

**val** quantity = df.select("quantity").rdd.cache().map{ row => row.getDouble(0) }

// 计算相关系数

**val** corr = Statistics.corr(pay, quantity)

println(s”The correlation is $corr”)

}

运行结果：The correlation is 0.9671352151112067

1.6集群的部署

Spark在生产环境中，主要部署在安装有Linux系统的集群中。在Linux系统中安装Spark需要预先安装JDK、Scala等所需的依赖。由于Spark是计算框架，所以需要预先在集群内搭建好存储数据的持久化层，这里安装的是HDFS。最后，我们就可以通过启动脚本运行程序了。这里以计算机scl为master节点，计算机sidxiong为slave节点来进行配置。在该例程中，我们使用的是spark的standalone部署模式。

系统要求：系统中的glibc版本要2.6或更高；系统中安装有JDK与SCALA。安装过程在之前的1.1已经有说，在这里就不重复了。

**配置SSH免密码登录**

集群模式需要用到SSH远程登录命令，Ubuntu默认安装了SSH client，此外，还需要安装SSH server：

$ sudo apt-get install openssh-server

安装完后，我们进入到终端窗口，利用 ssh-keygen 生成密钥，并将密钥加入到授权中

$ ssh-keygen -t rsa #生成密钥

$ cd .ssh/ #在该目录下，公钥在id\_rsa.pub中

我们将id\_rsa.pub文件中的字符拷贝到要登录的电脑上的该目录下的authorized\_keys文件中，即可实现免密码登录。注意：不论是主-从还是从-主，应该都要做到能免密码登录才行。

测试命令：$ ssh [主机名]

**安装hadoop**

安装hadoop的主要用途是在spark上能调用HDFS。首先，前往hadoop的官网选取一个镜像网址下载，这里下载的是2.6版本，得到一个hadoop-2.6.0.tar.gz压缩文件，解压到特定目录。注意：解压的目录在集群的任何计算机中都应当一致！在这里统一路径为/home/hpc/hadoop-2.6.0。进入hadoop的配置目录 /home/hpc/hadoop-2.6.0/etc/hadoop,打开文件hadoop-env.sh，在里面加入jdk路径：

JAVA\_HOME=/usr/lib/jvm/jdk1.7.0\_79（按照自己的实际安装位置）

保存并关闭之。

在/etc/hosts文件中映射好计算机名字与IP地址，例如：

192.168.1.63 sidxiong

192.168.1.32 scl

192.168.1.3 xen-centos1

192.168.1.4 xen-centos2

192.168.1.5 xen-centos3

进入hadoop的配置目录 /home/hpc/hadoop-2.6.0/etc/hadoop的配置目录，修改slaves、core-site.xml、hdfs-site.xml、mapred-site.xml、yarn-site.xml 这5个配置文件中的内容。

在文件slaves中，将原来的所有内容删除，往里面添加slaves节点的主机名，在这里只有一个slave节点，即sidxiong，所以改文件中只有一行内容：sidxiong 。如果有多个slave节点，则每行一个节点名称即可。

在文件 core-site.xml中，在xml结构中添加如下内容:

<property>

<name>fs.defaultFS</name>

<value>hdfs://scl:9000</value>

</property>

<property>

<name>hadoop.tmp.dir</name>

<value>file:/home/hpc/hadoop-2.6.0/tmp</value>

<description>Abase for other temporary directories.</description>

</property>

在文件 hdfs-site.xml中，只有一个Slave，所以dfs.replication的值设为1，在xml结构中添加如下内容：

<property>

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

<value>scl:50090</value>

</property>

<property>

<name>dfs.replication</name>

<value>1</value>

</property>

<property>

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

<value>file:/home/hpc/hadoop-2.6.0/tmp/dfs/name</value>

</property>

<property>

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

<value>file:/home/hpc/hadoop-2.6.0/tmp/dfs/data</value>

文件mapred-site.xml，这个文件不存在，首先需要从模板中复制一份，然后在xml结构中添加如下内容：

<property>

<name>mapreduce.framework.name</name>

<value>yarn</value>

</property>

文件 yarn-site.xml中，在xml结构中添加如下内容：

<property>

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

<value>scl</value>

</property>

<property>

<name>yarn.nodemanager.aux-services</name>

<value>mapreduce\_shuffle</value>

</property>

做好配置后，将整个hadoop的安装打包发送到slave节点上，然后在节点上解压到同样的目录下即可。（最好是先压缩后再通过scp命令发送）接下来回到主节点上，运行hadoop：

$ bin/hdfs namenode -format #首次运行需要格式化，之后运行不需要

$ sbin/start-dfs.sh

$ sbin/start-yarn.sh

通过jps命令查看各个节点所启动的进程，在主节点中应当包含以下几项：nameNode、SecondrryNameNode、ResourceManager；

而slave节点应当包含DataNode和NodeManager进程。

若从节点中没有DataNode，可以删除$HADOOP\_HOME目录下的tmp文件夹，再重新运行试试。

到这里，hadoop集群的安装就现告一段落了。

**安装Spark**

在安装完hadoop后，安装Spark就很轻松了，进入官网下载对应的Spark程序包，下载地址：spark.apache.org/downloads.html 当然，也可以通过wget的命令来下载。

下载后解压，在环境变量中配置如下：

SPARK\_HOME=/home/hpc/spark-1.3.0-bin-hadoop2.4

进入Spark安装路径，找到conf下的spark-env.sh（若没有则创建一个），在里面添加如下内容：

export SPARK\_MASTER\_IP=192.168.1.32

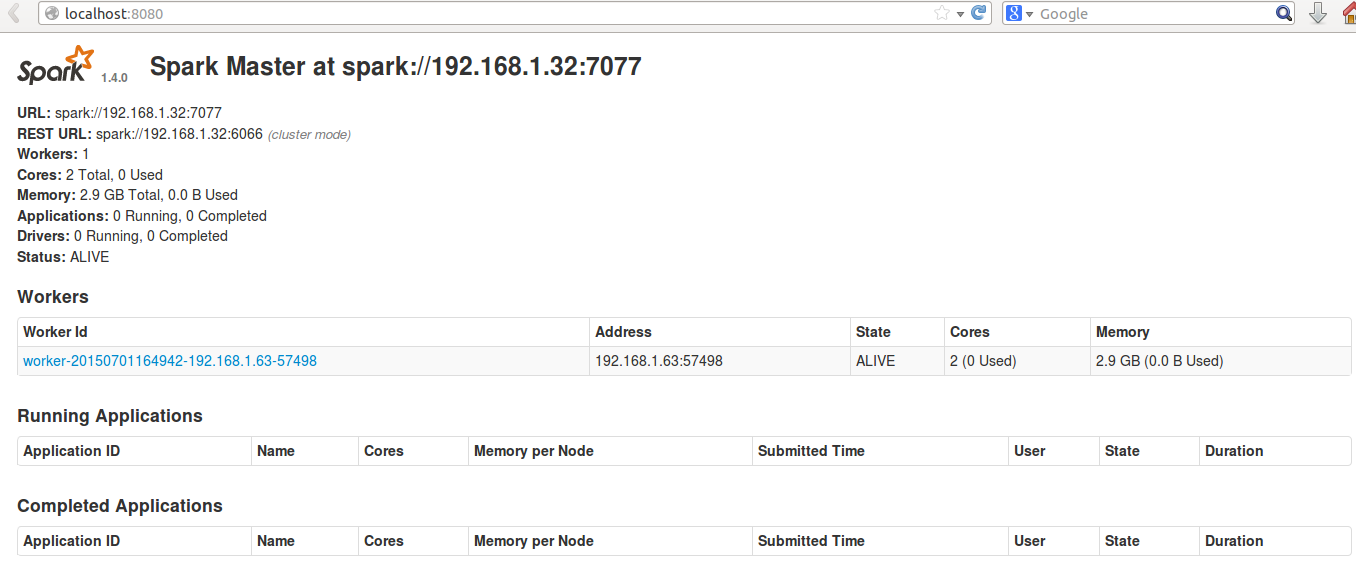
export MASTER=spark://192.168.1.32:7077

在slaves文件中添加一项内容，就是从节点的机器名，这里为sidxiong。

经过上述的简单配置后，将配置好的整个Spark路径打包发送到从节点，并解压到与主节点相同的路径。回到主节点，在spark的安装目录下执行：

$ ./sbin/start-all.sh

打开浏览器，进入localhost:8080应该就能看到主节点和从节点的运行情况了(若端口有冲突，则在刚才的spark-env.sh中相应的端口，例如SPARK\_MASTER\_WEBUI\_PORT)。打开网页的情况类似于下图:

若要停止，则用如下命令：

$ ./sbin/stop-all.sh

至此，Spark的安装也已完成。

**获取HDFS中的文件运行wordcount程序**

在主节点上，先将hadoop启动起来：

$ sbin/start-dfs.sh

$ sbin/start-yarn.sh

将本地要处理的文档复制到HDFS中：首先在桌面上新建一个test.txt，该文件中每行为一个单词，随便输入一些内容，然后进入hadoop的安装目录，执行以下命令：

$ bin/hdfs dfs -mkdir -p /user/hpc/test #创建目录

$ bin/hdfs dfs -put /home/hpc/Desktop/test.txt /user/hpc/test/

#将本地文件放入hdfs系统

$ bin/hdfs dfs -ls -R #查看文件



可以看到hdfs系统中出现了该文件。接下来，进入SPARK\_HOME，先启动SPARK，然后进spark-shell：

$ ./sbin/start-all.sh

$ bin/spark-shell

执行wordcount：

scala> val file=sc.textFile("hdfs://192.168.1.32:9000/user/hpc/test/test.txt")

scala> file.count() #执行结果为text.txt内的单词个数

**打成jar包执行wordcount程序**

可用sbt或是其他IDE打包，打包好之后应该会有一个.jar文件，该过程在此省略（通常使用Intellij，在这里不介绍）。

scala代码：

**package** wgzc.scala

**import** org.apache.spark.SparkContext

**import** org.apache.spark.SparkConf

**object** helloScala {

**def** main(args: Array[String]) {

**val** conf = new SparkConf().setAppName("WordConnt")

conf.setMaster("spark://192.168.1.32:7077")

**val** sc = new SparkContext(conf)

**val** lines = sc.textFile("hdfs://192.168.1.32:9000/user/hpc/test/test.txt")

**val** count = lines.count()

println("the count num is :" + count)

}

}

先启动Spark，之后在Spark目录下执行：

$ bin/spark-submit --class wgzc.scala.helloScala --master

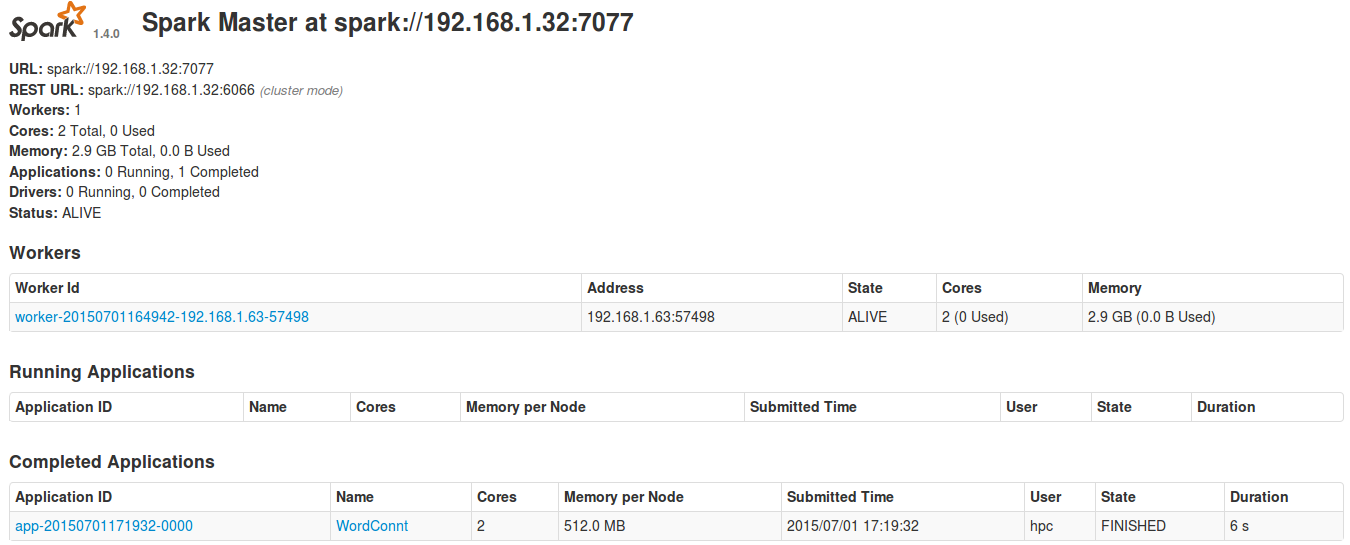
spark://192.168.1.32:7077

/home/hpc/IdeaProjects/T0623/out/artifacts/T0623\_jar/T0623.jar #jar包位置

执行结果应该同在shell中一样：

(the count num is : 10)

打开浏览器webUI，刷新页面，在Completed Applications下多了刚才执行的任务，如图：



1.7利用MLlib进行用户画像

原理介绍

协同过滤是推荐系统普遍的使用方法，该技术的本质目的是填充user-item关联矩阵中的缺失数据项。MLlib支持基于模型的协同过滤，通过一个隐含引资的消极和来预测缺失的数据项。MLlib通过实现交替最小二乘法（ALS）去求出这些隐含引资（latent factors）。ALS算法为了最小化损失函数，通过交替固定user与product向量，对损失函数求导，最终求出逼近Ratings矩阵的结果，并使用该结果进行商品推荐。

数据来源为单个加油站收集的购买记录数据，该数据有100多万条购买记录，由于该数据是用户购买的记录，用户的消费商品在通常情况下相关性并不大（例如用户买了一包烟和一瓶洗发水，而这是很正常的），所以计算结果差强人意，若是有了用户的行为表，通过用户点击商品的次数进行推荐，效果应该会比较好。

**部分代码及主要流程**

程序分为4个函数，根据输入的参数来决定如何运行，其中：

clearData函数的作用是预处理文本数据，剔除掉脏数据，保留有价值的数据；

run\_prc函数的作用是将统计结果和推荐结果以XML格式输出，方便查看数据

Purchase函数的作用是将统计消费结果插入数据库中

rec函数的作用是讲推荐结果update至数据库中

**if** (params(3) == "1") run\_prc(params) //以XML格式存入HDFS中

**else if** (params(3) == "2") Purchase(params) //消费统计

**else if** (params(3) == "3") rec(params) //推荐商品

**else if** (params(3) == "4") clearData(params) //预处理文本数据

**else** *println*("please choose modol")

**def** clearData(params: Array[String]) {…}

**def** run\_prc(params: Array[String]) {...}

**def** rec(params: Array[String]) {...}

**def** Purchase(params: Array[String]) {…}

详细代码及注释在src/zcdata文件夹下

**运行方法**

前往spark目录，将程序的jar包submit到集群中即可。

运行方法：在shell中前往SPARK\_HOME并执行如下命令：

hpc@scl:~/data/spark-1.4.0-bin-hadoop2.6$ ./bin/spark-submit --master spark://192.168.1.32:7077

--class zcdata.RecFromHDFS /home/hpc/IdeaProjects/recFromHDFS/out/artifacts/recFromHDFS\_jar/recFromHDFS.jar

其中，--master指定了master节点IP，--class是要运行的主类，最后一行是包所在的目录，第一次运行请在最后加上-h查看运行帮助。

如图所示：

