****

网络工程

企业开发实训报告

题目名称：基于Spark的WordCount实现

班 级：网络151班

学 号：201508030115

学生姓名：冯璐璐

指导教师：刘小明

2018 年 6 月

# 中文摘要

Apache Spark 是专为大规模数据处理而设计的快速通用的计算引擎。Spark是UC Berkeley AMP lab (加州大学伯克利分校的AMP实验室)所开源的类Hadoop MapReduce的通用并行框架，Spark，拥有Hadoop MapReduce所具有的优点；但不同于MapReduce的是——Job中间输出结果可以保存在内存中，从而不再需要读写HDFS，因此Spark能更好地适用于数据挖掘与机器学习等需要迭代的MapReduce的算法Spark 是一种与 Hadoop 相似的开源集群计算环境，但是两者之间还存在一些不同之处，这些有用的不同之处使 Spark 在某些工作负载方面表现得更加优越，换句话说，Spark 启用了内存分布数据集，除了能够提供交互式查询外，它还可以优化迭代工作负载。

关键词：Spark 大数据 Scala hadoop

[中文摘要 2](#_Toc364690821)

[第一章 Apache Spark介绍 4](#_Toc743302202)

[1.1 Spark的产生 4](#_Toc624336056)

[1.2 Spark的特点 4](#_Toc1594379870)

[1.3 Spark的性能特点 5](#_Toc1455201074)

[1.4 Spark的基本原理 6](#_Toc1397996533)

[1.5 计算方法 6](#_Toc265985166)

[第二章 Spark的架构 8](#_Toc1749873934)

[2.1 Spark的体系结构 8](#_Toc155276357)

[2.2 架构及生态 9](#_Toc1483016364)

[2.3 Spark的特性 12](#_Toc214906718)

[2.4 Spark与hadoop: 13](#_Toc1754856660)

[2.5 运行流程及特点 14](#_Toc470649370)

[第三章 Spark环境搭建 16](#_Toc1921931911)

[3.1 下载安装包 16](#_Toc1179779788)

[3.2搭建hadoop环境 16](#_Toc2060759210)

[3.3设置hadoop配置文件 18](#_Toc1934243026)

[3.4设置环境变量/etc/profile 22](#_Toc1821357011)

[3.5 配置hadoop节点信息 23](#_Toc893068662)

[3.6配置ssh无密码登录 24](#_Toc702442423)

[3.7配置master免密码登录slave1 25](#_Toc1968457417)

[3.8 Hadoop运行及检测 25](#_Toc550349954)

[3.8修改Spark配置文件 26](#_Toc739405869)

[3.9 Spark集群启动与检测 27](#_Toc172975266)

[3.10 集群启动效果预览 28](#_Toc1688861197)

[第四章 IDEA开发环境 30](#_Toc570842794)

[4.1 安装IDEA及插件 30](#_Toc1170323808)

[4.2 建立maven项目 31](#_Toc83307110)

[4.3 程序功能设计 34](#_Toc408297464)

## 第一章 Apache Spark介绍

### 1.1 Spark的产生

Apache Spark是专为大规模数据处理而设计的快速通用的计算引擎。现在形成一个高速发展应用广泛的生态系统。一个围绕速度、易用性和复杂分析构建的大数据处理框架，最初在2009年由加州大学伯克利分校的AMPLab开发，并于2010年成为Apache的开源项目之一，与Hadoop和Storm等其他大数据和MapReduce技术相比，Spark有如下优势：

（1）Spark提供了一个全面、统一的框架用于管理各种有着不同性质（文本数据、图表数据等）的数据集和数据源（批量数据或实时的流数据）的大数据处理的需求

（2）官方资料介绍Spark可以将Hadoop集群中的应用在内存中的运行速度提升100倍，甚至能够将应用在磁盘上的运行速度提升10倍

### 1.2 Spark的特点

Spark 主要有三个特点：

首先，高级 API 剥离了对集群本身的关注，Spark 应用开发者可以专注于应用所要做的计算本身。

其次，Spark 很快，支持交互式计算和复杂算法。

最后，Spark 是一个通用引擎，可用它来完成各种各样的运算，包括 SQL 查询、文本处理、机器学习等，而在 Spark 出现之前，我们一般需要学习各种各样的引擎来分别处理这些需求。

### 1.3 Spark的性能特点

（1）更快的速度

内存计算下，Spark 比 Hadoop 快100倍。

（2）易用性

Spark 提供了80多个高级运算符。

（3）通用性

Spark 提供了大量的库，包括SQL、DataFrames、MLlib、GraphX、Spark Streaming。 开发者可以在同一个应用程序中无缝组合使用这些库。

（4）支持多种资源管理器

Spark 支持 Hadoop YARN，Apache Mesos，及其自带的独立集群管理器

（5）Spark生态系统

Shark：Shark基本上就是在Spark的框架基础上提供和Hive一样的HiveQL命令接口，为了最大程度的保持和Hive的兼容性，Spark使用了Hive的API来实现query Parsing和 Logic Plan generation，最后的PhysicalPlan execution阶段用Spark代替HadoopMapReduce。通过配置Shark参数，Shark可以自动在内存中缓存特定的RDD，实现数据重用，进而加快特定数据集的检索。同时，Shark通过UDF用户自定义函数实现特定的数据分析学习算法，使得SQL数据查询和运算分析能结合在一起，最大化RDD的重复使用。

SparkR：SparkR是一个为R提供了轻量级的Spark前端的R包。 SparkR提供了一个分布式的data frame数据结构，解决了 R中的data frame只能在单机中使用的瓶颈，它和R中的data frame 一样支持许多操作，比如select,filter,aggregate等等。（类似dplyr包中的功能）这很好的解决了R的大数据级瓶颈问题。 SparkR也支持分布式的机器学习算法，比如使用MLib机器学习库。SparkR为Spark引入了R语言社区的活力，吸引了大量的数据科学家开始在Spark平台上直接开始数据分析之旅。

### 1.4 Spark的基本原理

Spark Streaming：构建在Spark上处理Stream数据的框架，基本的原理是将Stream数据分成小的时间片断（几秒），以类似batch批量处理的方式来处理这小部分数据。Spark Streaming构建在Spark上，一方面是因为Spark的低延迟执行引擎（100ms+），虽然比不上专门的流式数据处理软件，也可以用于实时计算，另一方面相比基于Record的其它处理框架（如Storm），一部分窄依赖的RDD数据集可以从源数据重新计算达到容错处理目的。此外小批量处理的方式使得它可以同时兼容批量和实时数据处理的逻辑和算法。方便了一些需要历史数据和实时数据联合分析的特定应用场合。

### 1.5 计算方法

Bagel: Pregel on Spark，可以用Spark进行图计算，这是个非常有用的小项目。Bagel自带了一个例子，实现了Google的PageRank算法。

当下Spark已不止步于实时计算，目标直指通用大数据处理平台，而终止Shark，开启SparkSQL或许已经初见端倪。

近几年来，大数据机器学习和数据挖掘的并行化算法研究成为大数据领域一个较为重要的研究热点。早几年国内外研究者和业界比较关注的是在 Hadoop 平台上的并行化算法设计。然而， HadoopMapReduce 平台由于网络和磁盘读写开销大，难以高效地实现需要大量迭代计算的机器学习并行化算法。随着 UC Berkeley AMPLab 推出的新一代大数据平台 Spark 系统的出现和逐步发展成熟，近年来国内外开始关注在 Spark 平台上如何实现各种机器学习和数据挖掘并行化算法设计。为了方便一般应用领域的数据分析人员使用所熟悉的 R 语言在 Spark 平台上完成数据分析，Spark 提供了一个称为 SparkR 的编程接口，使得一般应用领域的数据分析人员可以在 R 语言的环境里方便地使用 Spark 的并行化编程接口和强大计算能力。

## Spark的架构

### 2.1 Spark的体系结构

Spark体系架构包括如下三个主要组件：

* 数据存储
* API
* 管理框架

接下来让我们详细了解一下这些组件。

（1）数据存储：

Spark用HDFS文件系统存储数据。它可用于存储任何兼容于Hadoop的数据源，包括HDFS，HBase，Cassandra等。

（2）API：利用API，应用开发者可以用标准的API接口创建基于Spark的应用。Spark提供Scala，Java和Python三种程序设计语言的API。

下面是三种语言Spark API。

* [Scala API](http://spark.apache.org/docs/latest/api/scala/index.html" \l "org.apache.spark.package" \t "/home/fenglulu/Documents\\x/_blank)
* [Java](http://spark.apache.org/docs/latest/api/java/index.html" \t "/home/fenglulu/Documents\\x/_blank) API
* [Python](http://spark.apache.org/docs/latest/api/python/index.html" \t "/home/fenglulu/Documents\\x/_blank) API

（3）资源管理：

Spark既可以部署在一个单独的服务器也可以部署在像Mesos或YARN这样的分布式计算框架之上。

（4）弹性分布式数据集

[弹性分布式数据集](https://spark.apache.org/docs/latest/programming-guide.html" \l "resilient-distributed-datasets-rdds" \t "/home/fenglulu/Documents\\x/_blank)（基于Matei的[研究论文](https://www.cs.berkeley.edu/~matei/papers/2012/nsdi_spark.pdf" \t "/home/fenglulu/Documents\\x/_blank)）或RDD是Spark框架中的核心概念。可以将RDD视作数据库中的一张表。其中可以保存任何类型的数据。Spark将数据存储在不同分区上的RDD之中。

RDD可以帮助重新安排计算并优化数据处理过程。

此外，它还具有容错性，因为RDD知道如何重新创建和重新计算数据集。

RDD是不可变的。你可以用变换（Transformation）修改RDD，但是这个变换所返回的是一个全新的RDD，而原有的RDD仍然保持不变。

RDD支持两种类型的操作：

* 变换（Transformation）
* 行动（Action）

变换：[变换](https://spark.apache.org/docs/latest/programming-guide.html" \l "transformations" \t "/home/fenglulu/Documents\\x/_blank)的返回值是一个新的RDD集合，而不是单个值。调用一个变换方法，不会有任何求值计算，它只获取一个RDD作为参数，然后返回一个新的RDD。

变换函数包括：map，filter，flatMap，groupByKey，reduceByKey，aggregateByKey，pipe和coalesce。

行动：[行动](https://spark.apache.org/docs/latest/programming-guide.html" \l "actions" \t "/home/fenglulu/Documents\\x/_blank)操作计算并返回一个新的值。当在一个RDD对象上调用行动函数时，会在这一时刻计算全部的数据处理查询并返回结果值。

行动操作包括：reduce，collect，count，first，take，countByKey以及foreach。

### 2.2 架构及生态

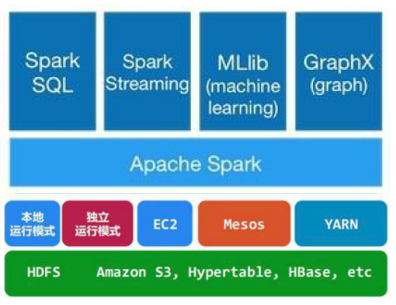
通常当需要处理的数据量超过了单机尺度(比如我们的计算机有4GB的内存，而我们需要处理100GB以上的数据)这时我们可以选择spark集群进行计算，有时我们可能需要处理的数据量并不大，但是计算很复杂，需要大量的时间，这时我们也可以选择利用spark集群强大的计算资源，并行化地计算，其架构示意图如下：

图2-1

Spark Core：包含Spark的基本功能；尤其是定义RDD的API、操作以及这两者上的动作。其他Spark的库都是构建在RDD和Spark Core之上的

Spark SQL：提供通过Apache Hive的SQL变体Hive查询语言（HiveQL）与Spark进行交互的API。每个数据库表被当做一个RDD，Spark SQL查询被转换为Spark操作。

Spark Streaming：对实时数据流进行处理和控制。Spark Streaming允许程序能够像普通RDD一样处理实时数据

MLlib：一个常用机器学习算法库，算法被实现为对RDD的Spark操作。这个库包含可扩展的学习算法，比如分类、回归等需要对大量数据集进行迭代的操作。

GraphX：控制图、并行图操作和计算的一组算法和工具的集合。GraphX扩展了RDD API，包含控制图、创建子图、访问路径上所有顶点的操作

Spark架构的组成图如下：

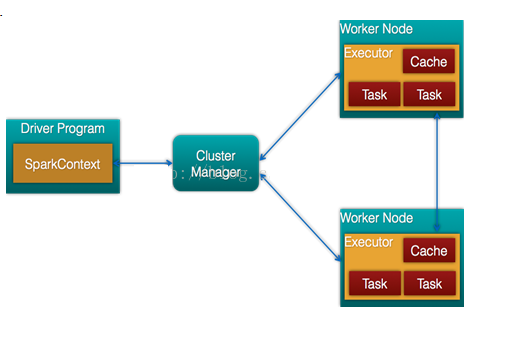


图2-2

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

Worker节点：从节点，负责控制计算节点，启动Executor或者Driver。

Driver： 运行Application 的main()函数

Executor：执行器，是为某个Application运行在worker node上的一个进程

除了这些库以外，还有一些其他的库，如BlinkDB和Tachyon。

[BlinkDB](http://blinkdb.org/" \t "/home/fenglulu/Documents\\x/_blank)是一个近似查询引擎，用于在海量数据上执行交互式SQL查询。BlinkDB可以通过牺牲数据精度来提升查询响应时间。通过在数据样本上执行查询并展示包含有意义的错误线注解的结果，操作大数据集合。

[Tachyon](http://tachyon-project.org/index.html" \t "/home/fenglulu/Documents\\x/_blank)是一个以内存为中心的分布式文件系统，能够提供内存级别速度的跨集群框架（如Spark和MapReduce）的可信文件共享。它将工作集文件缓存在内存中，从而避免到磁盘中加载需要经常读取的数据集。通过这一机制，不同的作业/查询和框架可以以内存级的速度访问缓存的文件。  
此外，还有一些用于与其他产品集成的适配器，如Cassandra（[Spark Cassandra 连接器](http://www.datastax.com/dev/blog/accessing-cassandra-from-spark-in-java" \t "/home/fenglulu/Documents\\x/_blank)）和R（SparkR）。Cassandra Connector可用于访问存储在Cassandra数据库中的数据并在这些数据上执行数据分析。

### 2.3 Spark的特性

（1）Spark通过在数据处理过程中成本更低的洗牌（Shuffle）方式，将MapReduce提升到一个更高的层次。利用内存数据存储和接近实时的处理能力，Spark比其他的大数据处理技术的性能要快很多倍。

(2）Spark还支持大数据查询的延迟计算，这可以帮助优化大数据处理流程中的处理步骤。Spark还提供高级的API以提升开发者的生产力，除此之外还为大数据解决方案提供一致的体系架构模型。

（3）Spark将中间结果保存在内存中而不是将其写入磁盘，当需要多次处理同一数据集时，这一点特别实用。Spark的设计初衷就是既可以在内存中又可以在磁盘上工作的执行引擎。当内存中的数据不适用时，Spark操作符就会执行外部操作。Spark可以用于处理大于集群内存容量总和的数据集。

Spark会尝试在内存中存储尽可能多的数据然后将其写入磁盘。它可以将某个数据集的一部分存入内存而剩余部分存入磁盘。开发者需要根据数据和用例评估对内存的需求。Spark的性能优势得益于这种内存中的数据存储。

### 2.4 Spark与hadoop:

Hadoop有两个核心模块，分布式存储模块HDFS和分布式计算模块Mapreduce

spark本身并没有提供分布式文件系统，因此spark的分析大多依赖于Hadoop的分布式文件系统HDFS

Hadoop的Mapreduce与spark都可以进行数据计算，而相比于Mapreduce，spark的速度更快并且提供的功能更加丰富

关系图如下：



图2-3

### 2.5 运行流程及特点

（1）spark运行流程图如下：

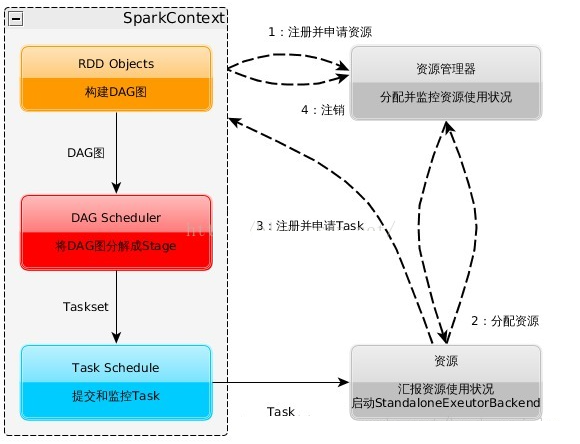


图2-4

构建Spark Application的运行环境，启动SparkContext

SparkContext向资源管理器（可以是Standalone，Mesos，Yarn）申请运行Executor资源，并启动StandaloneExecutorbackend，

Executor向SparkContext申请Task

SparkContext将应用程序分发给Executor

SparkContext构建成DAG图，将DAG图分解成Stage、将Taskset发送给Task Scheduler，最后由Task Scheduler将Task发送给Executor运行

Task在Executor上运行，运行完释放所有资源

（2） Spark运行特点：

每个Application获取专属的executor进程，该进程在Application期间一直驻留，并以多线程方式运行Task。这种Application隔离机制是有优势的，无论是从调度角度看（每个Driver调度他自己的任务），还是从运行角度看（来自不同Application的Task运行在不同JVM中），当然这样意味着Spark Application不能跨应用程序共享数据，除非将数据写入外部存储系统

Spark与资源管理器无关，只要能够获取executor进程，并能保持相互通信就可以了

提交SparkContext的Client应该靠近Worker节点（运行Executor的节点），最好是在同一个Rack里，因为Spark Application运行过程中SparkContext和Executor之间有大量的信息交换。Task采用了数据本地性和推测执行的优化机制

## 第三章 Spark环境搭建

### 3.1 下载安装包

环境配置版本如下：

* Spark-2.3.1
* Scala-2.11.12
* Hadoop-2.7.6

### 3.2搭建hadoop环境

1.因为Spark运行时还需要hadoop的hdfs文件系统，所以首先搭建hadoop环境

下载spark-2.3.1.tgz，[前往下载](http://spark.apache.org/downloads.html)

图3-1

选择spark-2.3.1-bin-hadoop2.7,该版本spark需要Scala2.11，和hadoop2.7+，所以我们还需要去[下载Scala2.11.12](https://scala-lang.org/download/2.11.12.html)和[hadoop2.7.6](http://mirrors.tuna.tsinghua.edu.cn/apache/hadoop/common/hadoop-2.7.6/)

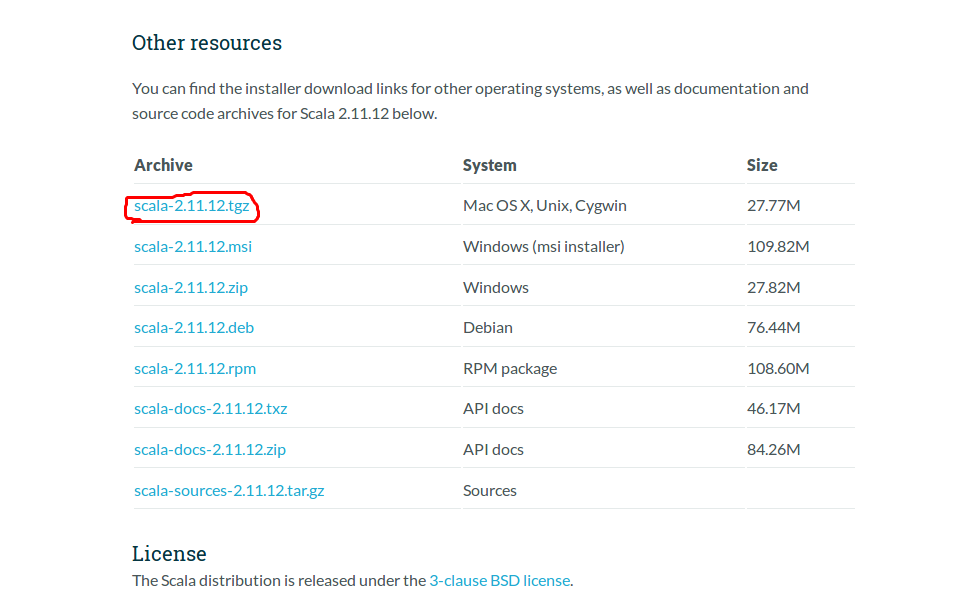


图3-2

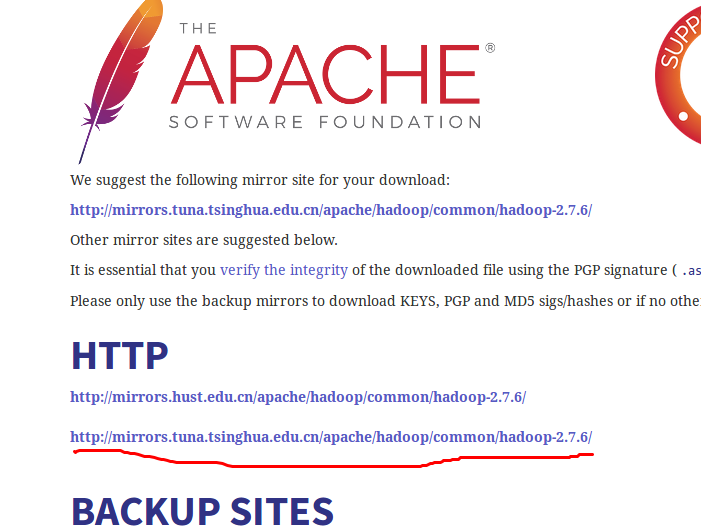


图3-3

2.将三个文件分别解压到/usr/spark文件夹下,先创建/usr/spark

mkdir /usr/spark

tar -zxvf scala-2.11.12.tgz -C /usr/spark/

tar -zxvf spark-2.3.1-bin-hadoop2.7.tgz -C /usr/spark/

tar -zxvf hadoop-2.7.6.tar.gz -C /usr/spark/

### 3.3设置hadoop配置文件

（1）新建以下目录

cd /usr/spark

sudo mkdir hadoop

fenglulu@fenglulu-Aspire-E5-572G:/usr/spark$ sudo mkdir hadoop/hdfs

fenglulu@fenglulu-Aspire-E5-572G:/usr/spark$ sudo mkdir hadoop/hdfs/data

fenglulu@fenglulu-Aspire-E5-572G:/usr/spark$ sudo mkdir hadoop/hdfs/name

fenglulu@fenglulu-Aspire-E5-572G:/usr/spark$ sudo mkdir hadoop/temp

(2)修改/usr/spark/hadoop-2.7.6/etc/hadoop/文件夹下的多个配置文件

a.修改core-site.xml

<configuration>

<property>

<name>fs.default.name</name>

<value>hdfs://master:9000</value>

<description>HDFS的URI，文件系统://namenode标识:端口号</description>

</property>

<property>

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

<value>file:/usr/spark/hadoop/temp</value>

<description>namenode上本地的hadoop临时文件夹</description>

</property>

</configuration>

b.修改hdfs-site.xml

<configuration>

<property>

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

<value>file:/usr/spark/hadoop/hdfs/name</value>

<description>namenode上存储hdfs名字空间元数据 </description>

</property>

<property>

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

<value>file:/usr/spark/hadoop/hdfs/data</value>

<description>datanode上数据块的物理存储位置</description>

</property>

<property>

<name>dfs.replication</name>

<value>1</value>

<description>副本个数，配置默认是3,应小于datanode机器数量</description>

</property>

<property>

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

<value>master:9001</value>

</property>

</configuration>

c.修改mapred-site.xml

<property>

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

<value>yarn</value>

</property>

<property>

<name>mapreduce.jobhistory.address</name>

<value>master:10020</value>

</property>

<property>

<name>mapreduce.jobhistory.webapp.address</name>

<value>master:19888</value>

</property>

d.修改yarn-site.xml

<configuration>

<!-- Site specific YARN configuration properties -->

<property>

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

<value>mapreduce\_shuffle</value>

</property>

<property>

<name>yarn.nodemanager.aux-services.mapreduce.shuffle.class</name>

<value>org.apache.hadoop.mapred.ShuffleHandler</value>

</property>

<property>

<name>yarn.resourcemanager.address</name>

<value>master:8032</value>

</property>

<property>

<name>yarn.resourcemanager.scheduler.address</name>

<value>master:8030</value>

</property>

<property>

<name>yarn.resourcemanager.resource-tracker.address</name>

<value>master:8031</value>

</property>

<property>

<name>yarn.resourcemanager.admin.address</name>

<value>master:8033</value>

</property>

<property>

<name>yarn.resourcemanager.webapp.address</name>

<value>master:8088</value>

</property>

</configuration>

e.接着将hadoop-env.sh,mapred-env.sh,yarn-env.sh中的JAVA\_HOME都修改为我们配置好的java路径（根据个人jdk设置而定）

### 3.4设置环境变量/etc/profile

sudo gedit /etc/profile

//在文件末尾追加如下代码，默认当前设备已设置了jdk1.8+；

#java environment

export JAVA\_HOME=/home/fenglulu/java/jdk1.8.0\_171

export CLASSPATH=.:$JAVA\_HOME/lib:$JRE\_HOME/lib:$CLASSPATH

export PATH=$JAVA\_HOME/bin:$JRE\_HOME/bin:$PATH

#maven environment

export M2\_HOME=/home/fenglulu/java/apache-maven-3.5.3

PATH=$M2\_HOME/bin:$PATH

#scala env

export SCALA\_HOME=/usr/spark/scala-2.11.12

export PATH=$PATH:$SCALA\_HOME/bin

#spark env

export SPARK\_HOME=/usr/spark/spark-2.3.1-bin-hadoop2.7

export PATH=$PATH:$SPARK\_HOME/bin

#hadoop env

export HADOOP\_HOME=/usr/spark/hadoop-2.7.6

export PATH=$PATH:HADOOP\_HOME/bin

注意：使用source /etc/profile 命令使其生效

3.5 配置hadoop节点信息  
修改/usr/spark/hadoop-2.7.6/etc/hadoop/下的slaves文件，添加节点，本次预计使用三个slave节点,内容如下：  
(2018-7-4更新：在一台电脑上配置两个节点，master与slave1.)

master

slave1

修改主机hosts文件，内容如下：

127.0.0.1 localhost

192.168.3.221 master

192.168.3.221 slave1

# The following lines are desirable for IPv6 capable hosts

::1 ip6-localhost ip6-loopback

fe00::0 ip6-localnet

ff00::0 ip6-mcastprefix

ff02::1 ip6-allnodes

ff02::2 ip6-allrouters

修改主机名称gedit /etc/hostname

gedit /etc/hostname

---

master

slave1

### 3.6配置ssh无密码登录

(1)ubuntu自带ssh-client，我们还需要ssh-server

apt-get install openssh-server

(2)启动ssh服务

/etc/init.d/ssh start

(3)查看sshd是否启动

ps -e | grep ssh

(4)root账户默认不允许登录ssh，修改权限

gedit /etc/ssh/sshd\_config

找到Authentication，修改PermitRootLogin yes，保存

/etc/init.d/ssh restart

(5)ssh登录本地

ssh localhost

SSH首次登录会有提示，直接输入yes即可，这时是需要密码的

(6)生成秘钥

ssh-keygen -t rsa

之后一直按Enter键，默认将秘钥保存在.shh/id\_rsa文件中

(7)RSA公钥加入授权文件

cd .ssh

cp id\_rsa.pub authorized\_keys

(8)重新登录，实现免密码登录localhost

### 3.7配置master免密码登录slave1

这里使用scp命令，可自行搜索相关信息，配置确保slave1节点已安装ssh-server

(1)将master根目录下密码复制到slave1的根目录下

root@master:~# scp ~/.ssh/id\_rsa.pub root@slave1:~/.ssh

(2)在slave1节点将RSA公钥加入授权文件

root@slave:~# cp .ssh/id\_rsa.pub authorized\_keys

如有多台计算机，重复以上操作即可

(3)在master节点上ssh登录slave1,实现免密码登录

ssh slave1

7、配置slave1节点的hadoop，同master节点，复制过去即可

### 3.8 Hadoop运行及检测

(1)格式化分布式文件系统，在master节点下

cd /usr/local/hadoop/hadoop-2.7.3

bin/hadoop namenode -format

(2)启动hadoop守护进程

sbin/start-all.sh

(3)检测启动情况

jps

这时在master节点可以看到NameNode，SecondaryNameNode，ResourceManager

在slave1节点可以看到DataNode，NodeManager，因为我也将master节地点添加为slaves，所以也能看到类似信息

(4)停止hadoop进程

sbin/stop-all.sh

### 3.8修改Spark配置文件

1)$SPARK\_HOME/conf/spark-env.sh cp spark-env.sh.template spark-env.sh  
添加以下内容:

export SCALA\_HOME=/usr/spark/scala-2.11.12

export JAVA\_HOME=/home/fenglulu/java/jdk1.8.0\_171

export SPARK\_MASTER\_IP=master

export SPARK\_WORKER\_MEMORY=512m

export HADOOP\_CONF\_DIR=$HADOOP\_HOME/etc/hadoop

2)$SPARK\_HOME/conf/slaves

cp slaves.template slaves

添加以下内容:

master

slave1

3.以上配置均在master节点进行，接下来将配置好的文件复制到slave节点，确保路径等均一致，不清楚的可以先看配置hadoop的文章

### 3.9 Spark集群启动与检测

1.启动spark集群

cd $SPARK\_HOME sbin/start-all.sh

2.查看

jps

3.结果  
master节点

root@master:/usr/spark/hadoop-2.7.6# jps

11826 DataNode

19794 Jps

19352 ResourceManager

11673 NameNode

19661 NodeManager

12078 SecondaryNameNode

slave1节点：

应该与master主机显示的一样，因为是单机上的伪分布式环境

root@master:/usr/spark/spark-2.3.1-bin-hadoop2.7# jps 16422 Jps 9238 SecondaryNameNode 16217 Master 6317 ResourceManager 5821 NameNode 16350 Worker

slave1节点：

6737 Worker

6774 Jps

4.关闭集群

`sbin/stop-all.sh`

### 3.10 集群启动效果预览

Hadoop的数据节点

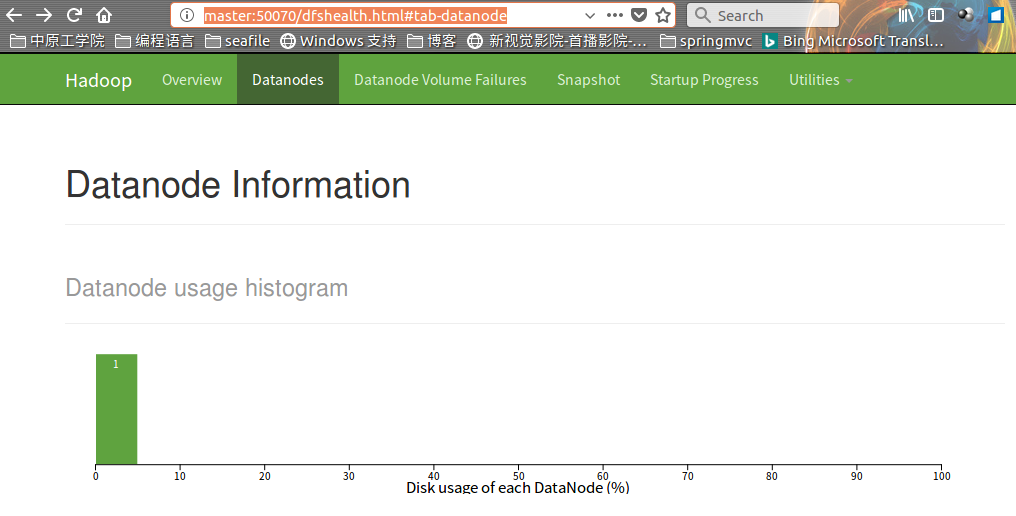
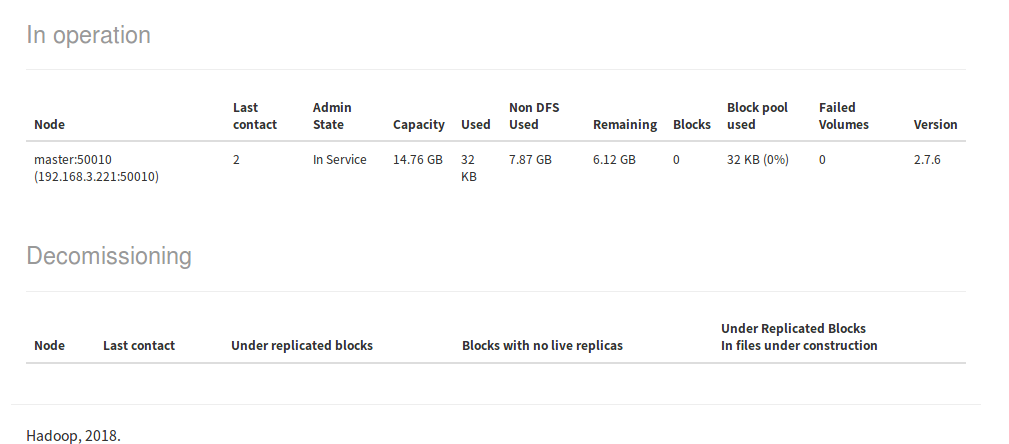


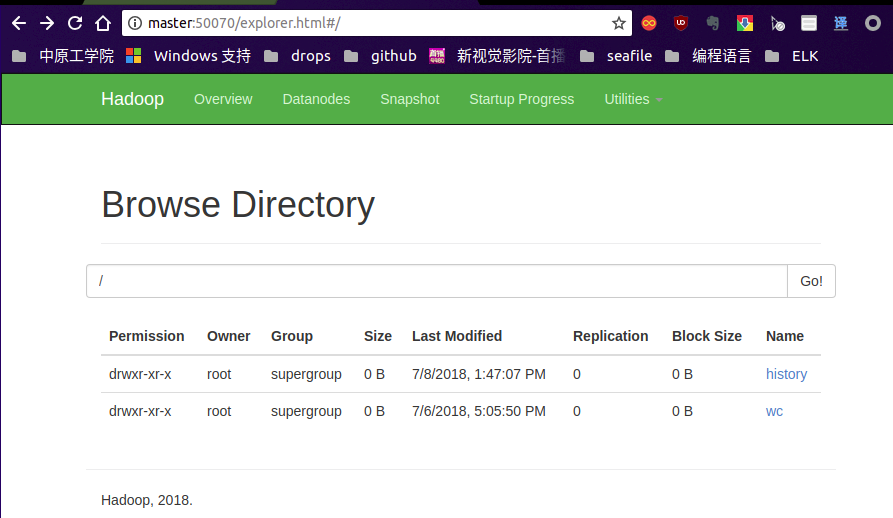
图3-4

节点信息

****

**图3-5**

**浏览文件系统**

****

**图3-5**

## 第四章 IDEA开发环境

### 4.1 安装IDEA及插件

下载idea for linux版本并在pligins中添加Scala插件，如下图：

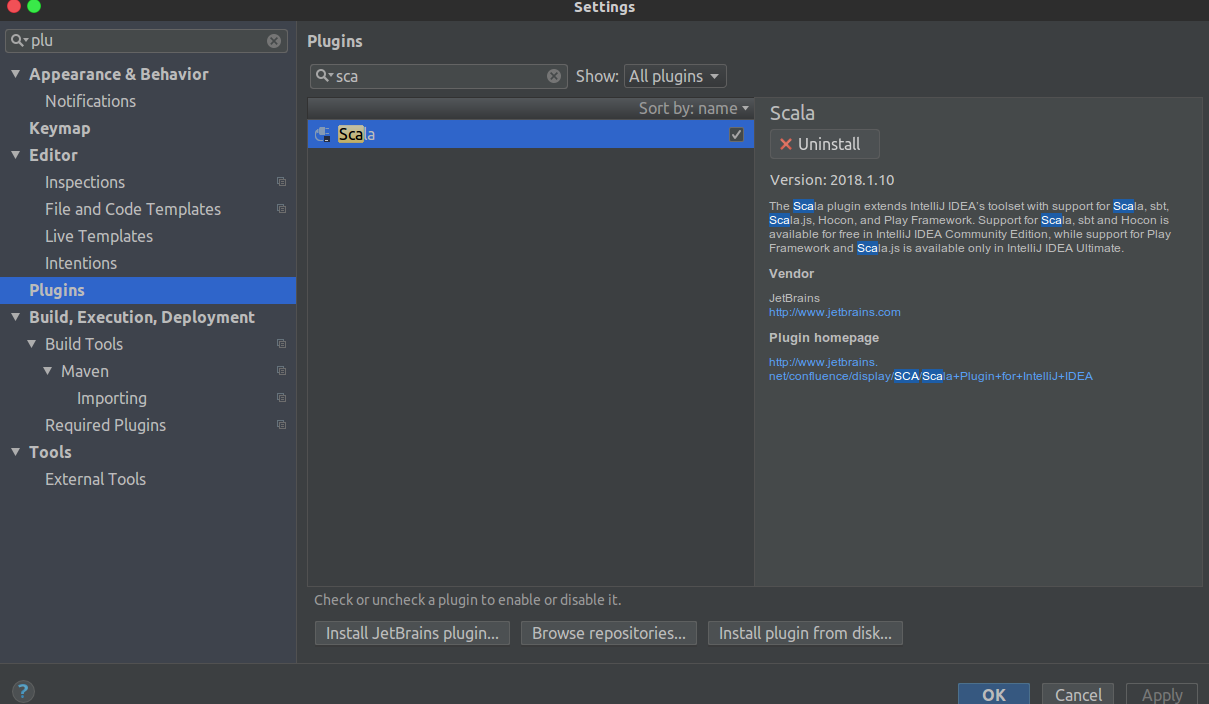


图4-1

### Selection_0094.2 建立maven项目

图4-2

在pom.xml中添加依赖：

<properties>

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

<spark.version>2.3.1</spark.version>

<scala.version>2.11</scala.version>

<hadoop.version>2.7.6</hadoop.version>

</properties>

<dependencies>

<dependency>

<groupId>org.ansj</groupId>

<artifactId>ansj\_seg</artifactId>

<version>5.1.1</version>

</dependency>

<dependency>

<groupId>org.apache.spark</groupId>

<artifactId>spark-core\_${scala.version}</artifactId>

<version>${spark.version}</version>

</dependency>

<dependency>

<groupId>org.apache.spark</groupId>

<artifactId>spark-sql\_${scala.version}</artifactId>

<version>${spark.version}</version>

</dependency>

<dependency>

<groupId>org.apache.spark</groupId>

<artifactId>spark-hive\_${scala.version}</artifactId>

<version>${spark.version}</version>

</dependency>

<dependency>

<groupId>org.apache.spark</groupId>

<artifactId>spark-streaming\_${scala.version}</artifactId>

<version>${spark.version}</version>

</dependency>

<dependency>

<groupId>org.apache.hadoop</groupId>

<artifactId>hadoop-client</artifactId>

<version>2.7.6</version>

</dependency>

<dependency>

<groupId>org.apache.spark</groupId>

<artifactId>spark-mllib\_${scala.version}</artifactId>

<version>${spark.version}</version>

</dependency>

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.12</version>

</dependency>

</dependencies>

**注意：所添加的依赖要与配置的集群环境安装包版本相符合**

### 4.3 程序功能设计

4.4 程序源码

import org.ansj.splitWord.analysis.BaseAnalysis

import org.apache.spark.graphx.Graph

import org.apache.spark.mllib.feature.{HashingTF, IDF}

import org.apache.spark.{SparkConf, SparkContext}

import org.apache.spark.mllib.linalg.{SparseVector => SV}

object groupNews {

def main(args: Array[String]): Unit = {

//相似度阈值

val sim = 0.5

val sparkconfig = new SparkConf().setAppName("DocSim").setMaster("local[2]")

val ctx = new SparkContext(sparkconfig)

val path = "/home/fenglulu/java/news/henanshifandaxue.txt"

//加载过滤数据(id,title,content)

val input = ctx.textFile(path).map(x => x.split("====").toSeq).filter(x => x.length == 3 && x(1).length > 1 && x(2).length > 1)

//分词(id,title,[words])

val splitWord = input.map(x => (x(0), x(1), BaseAnalysis.parse(x(2)).toStringWithOutNature(" ").split(" ").toSeq))

//聚类初始化 计算文章向量(id,(id,content,title))

val init\_rdd = input.map(a => {

(a(0).toLong, a)

})

init\_rdd.cache()

//计算TF-IDF特征值

val hashingTF = new HashingTF(Math.pow(2, 18).toInt)

//计算TF

val newSTF = splitWord.map(x => (x.\_1, hashingTF.transform(x.\_3))

)

newSTF.cache()

//构建idf model

val idf = new IDF().fit(newSTF.values)

//将tf向量转换成tf-idf向量

val newsIDF = newSTF.mapValues(v => idf.transform(v)).map(a => (a.\_1, a.\_2.toSparse))

newsIDF.take(10).foreach(x => println(x))

//构建hashmap索引 ,特征排序取前10个

val indexArray\_pairs = newsIDF.map(a => {

val indices = a.\_2.indices

val values = a.\_2.values

val result = indices.zip(values).sortBy(-\_.\_2).take(10).map(\_.\_1)

(a.\_1, result)

})

//(id,[特征ID])

indexArray\_pairs.cache()

indexArray\_pairs.take(10).foreach(x => println(x.\_1 + " " + x.\_2.toSeq))

//倒排序索引 (词ID,[文章ID])

val index\_idf\_pairs = indexArray\_pairs.flatMap(a => a.\_2.map(x => (x, a.\_1))).groupByKey()

index\_idf\_pairs.take(10).foreach(x => println(x.\_1 + " " + x.\_2.toSeq))

//倒排序

val b\_content = index\_idf\_pairs.collect.toMap

//广播全局变量

val b\_index\_idf\_pairs = ctx.broadcast(b\_content)

//广播TF-IDF特征

val b\_idf\_parirs = ctx.broadcast(newsIDF.collect.toMap)

//相似度计算 indexArray\_pairs(id,[特征ID]) b\_index\_idf\_pairs( 124971 CompactBuffer(21520885, 21520803, 21521903, 21521361, 21524603))

val docSims = indexArray\_pairs.flatMap(a => {

//将包含特征的所有文章ID

var ids: List[Long] = List()

//存放文章对应的特征

var idfs: List[(Long, SV)] = List()

//遍历特征，通过倒排序索引取包含特征的所有文章,除去自身

a.\_2.foreach(b => {

ids = ids ++ b\_index\_idf\_pairs.value.get(b).get.filter(x => (!x.equals(a.\_1))).map(x => x.toLong).toList

})

//b\_idf\_parirs(tf-idf特征),遍边文章，获取对应的TF-IDF特征

ids.foreach(b => {

idfs = idfs ++ List((b, b\_idf\_parirs.value.get(b.toString).get))

})

//获取当前文章TF-IDF特征

val sv1 = b\_idf\_parirs.value.get(a.\_1).get

import breeze.linalg.\_

//构建当前文章TF-IDF特征向量

val bsv1 = new SparseVector[Double](sv1.indices, sv1.values, sv1.size)

//遍历相关文章

val result = idfs.map {

case (id2, idf2) =>

val sv2 = idf2.asInstanceOf[SV]

//对应相关文章的特征向量

val bsv2 = new SparseVector[Double](sv2.indices, sv2.values, sv2.size)

//计算余弦值

val cosSim = bsv1.dot(bsv2) / (norm(bsv1) \* norm(bsv2))

(a.\_1, id2, cosSim)

}

// 文章1，文章2，相似度

result.filter(a => a.\_3 >= sim)

})

docSims.take(10).foreach(x => println(x))

//取出所有，有相似度的文章

val vertexrdd = docSims.map(a => {

(a.\_2.toLong, a.\_1.toLong)

})

//构建图

val graph = Graph.fromEdgeTuples(vertexrdd, 1)

val graphots = Graph.graphToGraphOps(graph).connectedComponents().vertices

//聚类初始化 计算文章向量 init\_rdd(id,(id,content,title))

init\_rdd.join(graphots).take(10).foreach(x => println(x))

val simrdd = init\_rdd.join(graphots).map(a => {

(a.\_2.\_2, (a.\_2.\_1, a.\_1))

})

val simrddtop = simrdd.groupByKey().filter(a => a.\_2.size >= 6).sortBy(-\_.\_2.size).take(50)

val simrdd2 = ctx.parallelize(simrddtop, 18)

simrdd2.take(10).foreach(x => {

val titles = x.\_2.map(x => x.\_1(1)).toArray

//选取事件主题名

val title = mostSimilartyTitle(titles)

println("事件---------------------" + title)

println(x.\_1)

x.\_2.foreach(x => println(x.\_2 + " " + x.\_1(0) + " " + x.\_1(1)))

})

}

/\*\*

\* 相似度比对 最短编辑距离

\* @param s

\* @param t

\* @return

\*/

def ld(s: String, t: String): Int = {

var sLen: Int = s.length

var tLen: Int = t.length

var cost: Int = 0

var d = Array.ofDim[Int](sLen + 1, tLen + 1)

var ch1: Char = 0

var ch2: Char = 0

if (sLen == 0)

tLen

if (tLen == 0)

sLen

for (i <- 0 to sLen) {

d(i)(0) = i

}

for (i <- 0 to tLen) {

d(0)(i) = i

}

for (i <- 1 to sLen) {

ch1 = s.charAt(i - 1)

for (j <- 1 to tLen) {

ch2 = t.charAt(j - 1)

if (ch1 == ch2) {

cost = 0

} else {

cost = 1

}

d(i)(j) = Math.min(Math.min(d(i - 1)(j) + 1, d(i)(j - 1) + 1), d(i - 1)(j - 1) + cost)

}

}

return d(sLen)(tLen)

}

/\*\*

\*

\* @param src

\* @param tar

\* @return

\*/

def similarity(src: String, tar: String): Double = {

val a: Int = ld(src, tar)

1 - a / (Math.max(src.length, tar.length) \* 1.0)

}

/\*\*

\* 选出一组字符串 中相似度最高的

\* @param strs

\* @return

\*/

def mostSimilartyTitle(strs: Array[String]): String = {

var map: Map[String, Double] = Map()

for (i <- 0 until strs.length) {

for (j <- i + 1 until strs.length) {

var similar = similarity(strs(i), strs(j))

if (map.contains(strs(i)))

map += (strs(i) -> (map.get(strs(i)).get + similar))

else

map += (strs(i) -> similar)

if (map.contains(strs(j)))

map += (strs(j) -> (map.get(strs(j)).get + similar))

else

map += (strs(j) -> similar)

}

} //end of for

if (map.size > 0)

map.toSeq.sortWith(\_.\_2 > \_.\_2)(0).\_1

else

""

}

}