# Spark安装与启动

环境：

Ubantu 14.04

Hadoop

Spark 2.4.0

## Hadoop安装

### Java安装

http://www.cnblogs.com/liugh/p/6623530.html

### SSH安装

<http://dblab.xmu.edu.cn/blog/install-hadoop/>

### Hadoop安装

http://dblab.xmu.edu.cn/blog/install-hadoop-cluster/

## Spark安装

http://dblab.xmu.edu.cn/blog/1187-2/

## Spark启动

http://dblab.xmu.edu.cn/blog/1187-2/

## 在HDFS分布式系统中管理文件

创建目录：./bin/hdfs dfs -mkdir -p /user/Hadoop

查看：

./bin/hdfs dfs -ls . 当前目录

./bin/hdfs dfs -ls /user/hadoop

./bin/hdfs dfs -ls / 根目录

上传文件：./bin/hdfs dfs -put /usr/local/spark/mycode/wordcount/word.txt /user/hadoop

# 数据导入

## 数据读取

### sc.textFile(path)

说明：

1. path可以是一个文件的路径，也可以是多个文件的路径，或者是一个目录
2. 返回结果是一个列表，列表中每个元素是文件中的一行数据
3. 列表中存储的数据应该是字符串？

样例：

1. sc.textFile("file:///usr/local/ README.md") //从本地系统读取文件
2. sc.textFile("hdfs://master:9000/examples/people.txt") //从分布式系统中读取文件

### sc.wholeTextFiles(dir\_path)

说明：

1. 读取一个目录下的所有文件
2. 返回的是一个列表，列表中每个元素都是一个(key,value)，key是文件路径，value是整个文件的内容，一个字符串

## 数据存储

### rdd.saveAsTextFile(dir\_path)

说明：

1. 将一个rdd的内容保存到dir\_path下的多个文件中，生成文件的个数由调用task的个数绝对
2. dir\_path一开始要不存在

代码样例：

1. 简单样例：

var rdd1 = sc.makeRDD(1 to 10,2)

rdd1.saveAsTextFile("hdfs://…/test/")

1. 只保存到一个文件中

rdd.coalesce(1,true).saveAsTextFile(dir\_path)

将数据从各个主机都收集到一起然后保存到一个文件中

### 存储后再读取

val textFile = sc.textFile("…/word.txt")

textFile.saveAsTextFile("…/writeback.txt")

val new\_textFile = sc.textFile("…/writeback.txt")

这样第二次读取之后new\_textFile中列表元素是一行行原数据还是(key,value)

# RDD操作

## RDD操作介绍

1. RDD上进行的操作主要分为两种：Transformations和Actions，前者不会触发计算，后者才会真正提交job进行计算。
2. RDD的计算结果也有两种形式，一是产生一个新的RDD，二是覆盖掉原来的RDD。

## RDD的惰性和持久化

### 惰性

RDD遇到Transformations操作并不会被执行，遇到Actions才会真正执行

常见Transformation API：

filter(func)、map(func)、flatMap(func)、groupByKey()、reduceByKey(func)

常见的Actions API:

count()、collect()、first()、take(n)、reduce(func)、foreach(func)

### 持久化

val list = List("Hadoop","Spark","Hive")

val rdd = sc.parallelize(list)

println(rdd.count())

println(rdd.collect().mkString(","))

上述代码遇到count()会触发一次从头到尾的计算，遇到collect()时又会触发一次从头到尾的运算，这会导致大量的重复性操作开销。

加入缓存机制：

val list = List("Hadoop","Spark","Hive")

val rdd = sc.parallelize(list)

rdd.cache() //等价于 rdd.persist()，有很多可选的存储参数

println(rdd.count())

println(rdd.collect().mkString(","))

使用cache()，rdd被生成之后就会被存在内存中，不需要重复计算。

当rdd不再被使用时，要及时使用unpersist()从内存中清除。

rdd.unpersist()

## foreach()、foreachpartition()

1. rdd.foreach(f)是对rdd中每个数据执行函数f定义的操作
2. rdd.foreachpartition(f)是对rdd中的每个块执行f操作，需要自己遍历块内的元素
3. 这两种方法没有返回值，不会产生新的rdd或者覆盖原来的rdd

代码样例：

def printData(x):

print x[0]

for line in x[1].split('\n'):

print line

dataRdd =sc.wholeTextFiles('hdfs://…')

dataRdd.foreach(printData)

## map()、flatMap()

1. rdd.map(f)对每个数据都执行f操作，返回一个新的rdd，新rdd中结果与原rdd一一对应，数据类型可以不同
2. rdd. flatMap(f)和map()类似，不过一个数据可以产生多个结果，即原rdd中的一个数据可以对应结果rdd中的多组数据

代码样例：

val a = sc.paralleize(1 to 4, 2) //生成rdd，第一个参数是一个列表，第二个是分块个数

val b = a.map(x=>x\*2)

val c = a.flatMap(x=>1 to x)

a.collect() //[1,2,3,4]

b.collect() //[1,4,9,16]

c.collect() //[1,1,2,1,2,3,1,2,3,4]

## filter()

1. rdd.filter(f)根据条件f从原rdd中选取满足要求的产生一个新的rdd

代码样例：

val inputRDD=sc.textFile("log.txt")

val errorRDD=inputRDD.filter(line=>line.contains("error"))

# 机器学习模型

Spark的机器学习库比较有限，不包含深度学习模型。

## 选择方案

### Elephas库

结合Keras和Spark，可以用Keras构建模型，然后将模型放入Spark中。

问题：

1. 很小众的库，中文文档几乎没有，而且都比较老
2. 版本限制较强(听说)

### 用文件系统做中转

将整个系统分为三个子系统：

1. 特征提取

使用Spark分布式处理数据，提取特征，然后写入分布式文件系统

1. 模型判别

从分布式文件系统中提出特征，单机进行模型判别，将判别结果再写入到分布式文件系统中

1. 后续处理

从分布式文件系统中提出判别结果，使用分布式的方式进行后续处理

问题：

1. 需要将我们的系统切分的比较明确，特征提取和模型判别分开
2. 三个系统之间如何进行调度？尤其是分布式系统与单机系统之间如何协调？资源如何分配？

### 能否将模型封装成一个普通函数？

使用rdd.map(f)，函数f里面加载模型，对数据进行判别，只用分布式进行测试不训练可不可以？

就算可行，每个块加载一次模型，也比较频繁了，浪费时间

## 模型数据格式

## 如何构建模型

## 能否使用非Spark的机器学习库

## 模型训练与测试

## 模型存储与加载

# 结果存储与获取

## 结果存储的形式?分布式数据库？分布式文件系统

## 其他程序如何获取分布式系统中的结果