# 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/

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

首先在每台机子上设置无密登录：

cd ~/.ssh/ # 若没有该目录，请先执行一次ssh localhost

ssh-keygen -t rsa # 会有提示，都按回车就可以

cat ./id\_rsa.pub >> ./authorized\_keys # 加入授权

然后在主节点上将秘钥发给各子节点(主节点上不要再重新执行上述设置无密操作的步骤，都在会出现问题)

scp ~/.ssh/id\_rsa.pub hadoop@Slave1:/home/hadoop/

scp ~/.ssh/id\_rsa.pub hadoop@Slave2:/home/hadoop/

在各子节点上将主节点秘钥加入授权

cat ~/id\_rsa.pub >> ~/.ssh/authorized\_keys

rm ~/id\_rsa.pub

### 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：/usr/local/hadoop/bin/hdfs dfs -mkdir -p /user/hadoop

查看：

/usr/local/hadoop/bin/hdfs dfs -ls . 当前目录

/usr/local/hadoop/bin/hdfs dfs -ls /user/hadoop

/usr/local/hadoop/bin/hdfs dfs -ls / 根目录

上传文件：

/usr/local/hadoop/bin/hdfs dfs –put /usr/local/spark/mycode/wordcount/word.txt /user/Hadoop

常用参数：

[-cat [-ignoreCrc] <src> ...]

[-copyFromLocal [-f] [-p] [-l] [-d] <localsrc> ... <dst>]

[-copyToLocal [-f] [-p] [-ignoreCrc] [-crc] <src> ... <localdst>]

[-count [-q] [-h] [-v] [-t [<storage type>]] [-u] [-x] <path> ...]

[-cp [-f] [-p | -p[topax]] [-d] <src> ... <dst>]

[-df [-h] [<path> ...]]

[-du [-s] [-h] [-x] <path> ...]

[-find <path> ... <expression> ...]

[-get [-f] [-p] [-ignoreCrc] [-crc] <src> ... <localdst>]

[-help [cmd ...]]

[-ls [-C] [-d] [-h] [-q] [-R] [-t] [-S] [-r] [-u] [<path> ...]]

[-mkdir [-p] <path> ...]

[-moveFromLocal <localsrc> ... <dst>]

[-moveToLocal <src> <localdst>]

[-mv <src> ... <dst>]

[-put [-f] [-p] [-l] [-d] <localsrc> ... <dst>]

[-rm [-f] [-r|-R] [-skipTrash] [-safely] <src> ...]

[-rmdir [--ignore-fail-on-non-empty] <dir> ...]

## 问题

### 虚拟机每次启动都会变IP，需要重新配置

sudo vim /etc/hosts

vim /usr/local/spark/conf/spark-env.sh

### Ssh无密登录问题

不能生成两次key，否则只能全删重新装

### 无密登录失效

主节点执行 ssh-add

### Java 1.7版本过低

### 关于hadoop在./start-all.sh指令启动后子节点没有datanode进程的解决办法

<https://blog.csdn.net/Islotus/article/details/78357857>

注意/usr/local/hadoop下文件权限问题，不要用sudo创建目录

# Spark下mongodb的使用

## Mongodb安装与启动

### 安装

<http://www.runoob.com/mongodb/mongodb-tutorial.html>

### 配置文件

默认情况下是没有配置文件的，我们可以自己写一个，然后在启动时加载该配置文件。

https://www.cnblogs.com/loswing/p/5012388.html

<https://www.cnblogs.com/cwp-bg/p/9479945.html>

比较重要的配置项：

1. bind\_ip

默认是localhost，只能通过本机访问。如果想其他主机访问，需要将bind\_ip改为当前主机的IP。

### 启动

先开一个shell执行 mongod，启动mongodb服务，也可以用nohup让其在后台执行。

### 连接

本地连接直接在shell中输入 mongo

## Spark连接配置

需要使用MongoDB Spark Connector才能连接这Spark与mangodb

### 使用shell模式

./pyspark --conf "spark.mongodb.input.uri=mongodb://127.0.0.1/test.people"

--conf "spark.mongodb.output.uri=mongodb://127.0.0.1/test.test"

--packages org.mongodb.spark:mongo-spark-connector\_2.11:2.4.0

指定packages之后他就会自己去下载，2.11是scala版本，2.4.0是spark版本

下载目录为：/home/audr/.ivy2/jars/，有两个jar文件

org.mongodb\_mongo-java-driver-3.9.0.jar

org.mongodb.spark\_mongo-spark-connector\_2.11-2.4.0.jar

### 在py文件里使用

将/home/audr/.ivy2/jars/下的文件全放到/usr/local/spark/jars/目录下，这样spark程序就可以使用相应的依赖包。

## 使用

### 创建session，加载配置

from pyspark.sql import SparkSession

#创建session

my\_spark = SparkSession \

.builder \

.appName("myApp") \

.config("spark.mongodb.input.uri", "mongodb://input\_ip/test.input") \

.config("spark.mongodb.output.uri", "mongodb://output\_ip/test.output") \

.getOrCreate()

### 读取数据

# 从默认配置中指定的input db中读取数据

df = my\_spark.read.format("com.mongodb.spark.sql.DefaultSource").load()

# 重新指定数据库uri

df=my\_spark.read.format("com.mongodb.spark.sql.DefaultSource").option("uri","mongodb://new\_ip/people.contacts").load()

# 按照限制读取数据

# 这里限制条件的写法和普通mangodb操作的写法相同

# age == 100

pipeline = "{'$match': {'age': 100}}"

df = my\_spark.read.format("com.mongodb.spark.sql.DefaultSource").option("pipeline", pipeline).load()

# 70 < age <= 90

pipeline = "{'$match': {'age': { $gt : 70, $lte : 90 }}}"

df = my\_spark.read.format("com.mongodb.spark.sql.DefaultSource").option("pipeline", pipeline).load()

### 存储数据

写入数据库有4种模式：overwrite ignore errorifexists append

overwrite : 先删除mongodb中指定的表，然后把数据写到这个表中

ignore : 如果mongodb中有这个表，就不写数据了，且不会报错

errorifexists : 如果mongodb中存在这个表就报错，如果不存在就正常写入

append : 不管mongodb中这个表存不存在直接往里写数据

new\_data = my\_spark.createDataFrame([("Bilbo Baggins", 50), ("Gandalf", 1000), ("Thorin", 195)], ["name", "age"]) # 第一个参数是数据，第二个是字段名

new\_data.write.format("com.mongodb.spark.sql.DefaultSource").mode("append").save()

# 数据导入

## RDD与dataframe之间的转换

## 数据读取

读取本地文件必须要求所有节点的本地同一路径都有这个文件才可以

### 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"))

# PySpark

## Python版本与环境变量

Python2.6以上

环境变量：<http://dblab.xmu.edu.cn/blog/1689-2/>

export PYTHONPATH=$SPARK\_HOME/python:$SPARK\_HOME/python/lib/py4j-0.10.4-src.zip:$PYTHONPATH

export PYSPARK\_PYTHON=python3 /python2.7

PYTHONPATH是将pyspark加入到python的搜索路径中

红色部分根据spark版本会变动，具体去相应目录下看

## 启动PyShark进行分布式编程

cd /usr/local/spark

./bin/pyspark --master local[\*] //本地运行，使用所有CPU

./bin/pyspark –master spark://master:7077 //连接集群spark://master:7077进行编程

## 运行python程序

API文档：

http://spark.apache.org/docs/0.7.3/api/pyspark/pyspark.context.SparkContext-class.html

程序样例1：

from pyspark import SparkContext

sc = SparkContext( 'spark://master:7077', 'test')

logFile = "file:///usr/local/spark/README.md"

logData = sc.textFile(logFile, 2).cache()

numAs = logData.filter(lambda line: 'a' in line).count()

numBs = logData.filter(lambda line: 'b' in line).count()

print('Lines with a: %s, Lines with b: %s' % (numAs, numBs))

说明：

1. 使用pyspark时会根据启动命令行自动生成sc，但写程序时要自己生成

## 文件路径问题

1. 对于各子节点，import会去它自己的搜索路径中查找相应的py文件，所以第三方库或者自己写的代码文件，都需要使用sc.addPyFile(a.py)将它们放到搜索路径下，这时使用import a.py就可以访问到。
2. SparkFiles.getRootDirectory()可以查看addPyFile()、addFile()上传文件的路径
3. sc.addPyFile()上传的文件不能重名，因为它们都在同一目录下。
4. 如果上传的a.py存在多层嵌套调用，无法使用import a.py调用的话，可以使用a = \_\_import\_\_(‘a’)调用

**重要：与job相关的代码，比如a.py，使用sc.addPyFile(a.py)上传到分布式中，那么a.py中只能在函数内import所需模块，不能将import放在模块外，否则会报错**

## 代码打包上传问题

main.py

dns/

A/

a.py

x.py

main.py里调用x.py，x.py里的函数调用a.py

所以应该将整个dns文件夹上传到分布式系统，同时将dns的目录加入到搜索路径

1. 首先打包：zip –r dns.zip dns
2. 上传sc.addPyFile(‘./dns.zip’)
3. 使用：

Main.py

from dns import x

rdd.map(x.method)

x.py

def method(data):

from A import a

…

# Spark多进程提交任务

# 问题

## textFile()第二个参数的设置有没有什么需要注意的地方？

## Python生成(key,value)这种rdd要怎么做？使用tuble还是dict?

## RDD什么时候会删除？执行完操作之后，程序还没结束，RDD会不会被删除？

## 对RDD进行的操作函数能够使用静态变量吗？如果静态变量是一个模型能成功使用吗？

## 分布式资源使用情况，需要设置什么参数

## 能同时运行多个分布式程序吗？

## StreamingContext监听文件时，如果文件还没有完全写入成功，这时会发生什么？