电子科技大学信息与软件工程学院

**实 验 报 告**

学 号 2019091604016

姓 名 夏野

（实验） 课程名称 大数据分析与智能计算

实验时间 2021-11-7

实验教师 罗瑜

**电 子 科 技 大 学**

**实 验 报 告**

1. **实验名称：**

Hadoop 和 Spark 安装配置

1. **实验学时：**

4学时

1. **实验目的：**
2. 掌握Hadoop的环境搭建；
3. 掌握Hadoop、Linux的基本命令。
4. **实验原理：**
5. Hadoop是一个由Apache基金会所开发的[分布式系统](https://baike.baidu.com/item/%E5%88%86%E5%B8%83%E5%BC%8F%E7%B3%BB%E7%BB%9F/4905336" \t "/Users/jackieshi/Documents\\x/_blank)基础架构。用户可以在不了解分布式底层细节的情况下，开发分布式程序。充分利用集群的威力进行高速运算和存储。Hadoop实现了一个[分布式文件系统](https://baike.baidu.com/item/%E5%88%86%E5%B8%83%E5%BC%8F%E6%96%87%E4%BB%B6%E7%B3%BB%E7%BB%9F/1250388" \t "/Users/jackieshi/Documents\\x/_blank)（Distributed File System），其中一个组件是[HDFS](https://baike.baidu.com/item/HDFS/4836121" \t "/Users/jackieshi/Documents\\x/_blank)（Hadoop Distributed File System）。HDFS有高[容错性](https://baike.baidu.com/item/%E5%AE%B9%E9%94%99%E6%80%A7/9131391" \t "/Users/jackieshi/Documents\\x/_blank)的特点，并且设计用来部署在低廉的（low-cost）硬件上；而且它提供高吞吐量（high throughput）来访问[应用程序](https://baike.baidu.com/item/%E5%BA%94%E7%94%A8%E7%A8%8B%E5%BA%8F/5985445" \t "/Users/jackieshi/Documents\\x/_blank)的数据，适合那些有着超大数据集（large data set）的应用程序。HDFS放宽了（relax）[POSIX](https://baike.baidu.com/item/POSIX/3792413)的要求，可以以流的形式访问（streaming access）文件系统中的数据。Hadoop的框架最核心的设计就是：[HDFS](https://baike.baidu.com/item/HDFS/4836121" \t "/Users/jackieshi/Documents\\x/_blank)和[MapReduce](https://baike.baidu.com/item/MapReduce/133425" \t "/Users/jackieshi/Documents\\x/_blank)。HDFS为海量的数据提供了存储，而MapReduce则为海量的数据提供了计算。
6. Spark是基于内存计算的大数据并行计算框架.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部署在大量的廉价硬件之上,形成集群。Spark是MapReduce的替代方案，而且兼容HDFS、Hive等分布式存储层，可融入Hadoop的生态系统，以弥补缺失MapReduce的不足。
7. **实验内容：**
8. Hadoop 单机模式安装；
9. Spark 安装；
10. 测试安装。
11. **实验器材（设备、元器件）：**
12. 软件环境：Ubuntu 20.04、jdk-8u181-linux-x64.tar、Hadoop 2.7.3、spark-3.0.1-bin-hadoop3.2.tgz、Parallels Desktop
13. 系统环境：MacOS
14. **实验步骤：**
15. 创建用户；
16. 安装ssh；
17. 安装Java环境；
18. 安装Hadoop；
19. 测试。
20. **实验结果与分析（含重要数据结果分析或核心代码流程分析）**
21. Hadoop 安装配置
22. 添加用户及用户组

创建 hadoop 用户和 hadoop 用户组，将 /hadoop 目录作为该用户的 HOME 目录。创建结果如图 1.1 所示:

|  |
| --- |
| Text  Description automatically generated图 1.1 添加用户结果 |

1. 添加 sudo 权限

接着，使用如下命令为新用户添加管理员权限，以免之后出现权限问题。

|  |
| --- |
| $ sudo usermod -G sudo hadoop |

创建结果如图 1.2 所示。

|  |
| --- |
| 图 1.2 添加权限 |

1. 安装及配置依赖的软件包
   1. Hadoop环境需要原装openssh-server, java等，这些软件包在实验环境中如果没有，就需要手工安装.
      1. 安装并启动open-ssh-server

|  |
| --- |
| $ sudo apt-get install openssh-server -y  $ sudo service ssh start |

安装结果如图1.3所示：

|  |
| --- |
| Text  Description automatically generated  图 1.3 安装结果 |

执行以下命令验证环境，结果如图 1.4 所示。

|  |  |
| --- | --- |
| $ sudo -V | |
| 图 1.4 ssh 服务启动结果 |

* 1. 配置 ssh 免密码登录

在 /hadoop 目录下执行以下命令，配置 ssh 环境免密码登录，结果如图 1.5 所示。

|  |  |
| --- | --- |
| $ ssh-keygen -t rsa  $ cat .ssh/id\_rsa.pub >> .ssh/authorized\_keys  $ chmod 600 .ssh/authorized\_keys  $ ssh localhost  $ su - hadoop | |
| 验证ssh免密码配置成功如图所示  Text  Description automatically generated  图 1.5 验证 ssh 免密码配置 |

验证登陆本机是否还需要密码：

|  |
| --- |
| Text  Description automatically generated  图 1.6 登陆结果 |

1. 下载并安装 JAVA 和 Hadoop
   1. 复制压缩包到/hadoop 目录下

使用 cp 命令将本地下载好的安装包复制到 /hadoop 目录下，结果如图 1.4 所示。

|  |
| --- |
| Text  Description automatically generated  图 1.4 复制压缩包结果 |

* 1. 解压并安装

使用如下命令，对 Hadoop 和 Java 的安装包解压缩并安装。

|  |
| --- |
| $ cd /hadoop  $ tar -zxvf hadoop-2.7.3.tar.gz  $ chmod 755 /hadoop  $ tar -zxvf jdk-8u181-linux-x64.tar.gz |

解包结果如图1.5所示：

|  |
| --- |
| Text  Description automatically generated  图 1.5 hadoop目录 |

* 1. 配置 Hadoop 环境变量

向 Hadoop 的配置文件中添加配置内容，如图 1.6 所示。配置成功后，使用如下命令激活新添加的环境变量。

|  |  |
| --- | --- |
| $ vi /hadoop/.bash\_profile  $ source ~/.bash\_profile | |
| 成功激活环境截图如下图所示  Text  Description automatically generated  图 1.6 激活环境 |

* 1. 查看 JAVA 和 Hadoop 版本

查看 Java 和 Hadoop 的版本号，结果如图 1.7 所示。

|  |
| --- |
| Text  Description automatically generated  图 1.7 版本查看结果 |

Hadoop 单机模式安装完成！

1. 实验结果

创建输入数据，采用 /etc/protocols 文件作为测试，执行 Hadoop 内置程序WordCount，统计词频。查看生成的单词统计数据，结果如图 1.8 所示。

|  |
| --- |
| Text  Description automatically generated  图 1.8 WordCount 单词统计结果 |

1. 作业练习

首先，我们使用rm命令删去上一次执行程序生成的 output 目录，然后使用 cp 命令复制 /var/log/dpkg.log 文件，并重新命名为 input。

|  |
| --- |
| $ sudo rm -rf output  $ cp /var/log/dpkg.log input |

执行 Hadoop 内置程序WordCount，统计词频，查看生成的单词统计数据，结果如图 1.9 所示。

|  |  |
| --- | --- |
| $/hadoop/hadoop-3.1.4/bin/Hadoop jar /hadoop/hadoop-3.1.4/share/hadoop/mapreduce/sources/hadoop-mapreduce-examples-3.1.4-sources.jar org.apache.hadoop.examples.WordCount input output | |
| Text  Description automatically generated  图 1.9 作业练习 |

1. Spark安装配置
2. Hadoop 伪分布式配置
   1. 配置 core-site.xml

|  |
| --- |
| Text  Description automatically generated  图 2.1配置 core-site.xml 结果 |

* 1. 配置 hdfs-site.xml

|  |
| --- |
| Text  Description automatically generated  图 2.2 配置 hdfs-site.xml 结果 |

* 1. 配置 hadoop-env.sh

打开 hadoop-env.sh 文件，修改 JAVA\_HOME 变量为全局的 JAVA\_HOME 变量。

|  |
| --- |
| Text  Description automatically generated  图 2.3 修改 JAVA\_HOME 变量 |

* 1. 格式化 namenode 节点

|  |
| --- |
| Text  Description automatically generated  图 2.4 格式化 namenode 结果 |

1. 启动 namenode 节点

使用./start-all.sh命令启动结点，使用如下命令检查 namenode 节点是否运行成功，可以看到 Hadoop 正在运行的进程，如图 2.5 所示。

|  |
| --- |
| $ ./start-all.sh  $ jps |

|  |
| --- |
| Text  Description automatically generated  图 2.5 查看启动的 namenode 节点 |

1. 安装与配置 Spark
2. 解压并安装 Spark

将 Spark 安装在 /hadoop 目录下。使用以下命令下载安装包并解压安装。

|  |
| --- |
| $ cd /hadoop  $wget http://archive.apache.org/dist/spark/spark-3.1.2/spark-3.1.2-bin-hadoop3.2.tgz $ tar -zxvf spark-3.0.1-bin-hadoop3.2.tgz |

解压后文件的位置是/hadoop/app：

|  |
| --- |
| 图 3.1 hadoop目录下的文件 |

1. 配置 Hadoop 环境变量

在 Yarn 上运行 Spark 需要配置 HADOOP\_CONF\_DIR、 YARN\_CONF\_DIR 和 HDFS\_CONF\_DIR 环境变量。打开配置文件，导入配置，并更新配置文件，结果如图 3.2 所示。

|  |
| --- |
| Text  Description automatically generated  图 3.2 配置环境变量 |

1. 修改配置文件

配置文件 spark-env.sh，结果如图 3.3 所示。

|  |
| --- |
| Text  Description automatically generated  图 3.3 配置文件 |

1. Spark 的启动
   1. 进入 spark-shell

启动 spark-shell，进入 Spark 的 shell 界面, 结果如图 3.4 所示。注意在启动的时候可能会遇到报错，是关于无效URL的问题，我们需要在bash\_profile配置文件里面加上export SPARK\_LOCAL\_HOSTNAME=localhost，告诉机器我们的spark地址是什么，之后就能够正常运行了。

|  |
| --- |
| Text  Description automatically generated  图 3.4 spark-shell 启动 |

* 1. 启动 Master

使用 jps 命令查看 Master 进程是否启动成功，结果如图 3.5 所示。

|  |
| --- |
| Text  Description automatically generated  图 3.5 Master 启动结果 |

* 1. 启动 Slave

使用 jps 命令查看 Worker 进程是否启动成功，结果如图 3.6 所示。

|  |
| --- |
| Text  Description automatically generated  图 3.6 Worker 启动结果 |

1. 验证 Spark

按照实验指导书运行 的实例，并且使用 grep 命令尝试对运行结果进行过滤，运行结果如图3.7所示。

|  |
| --- |
| Text  Description automatically generated  图 3.7 搜索运行结果 |

1. **总结及心得体会：**

实验一的主要内容是在Ubuntu操作系统下搭建属于自己的hadoop环境，并使用hadoop提供的一些默认样例程序跑一些代码体验hadoop的使用。同时安装并配置了Spark环境。我认为实验一更多是为后面的实验做一些预先准备，例如搭建后续实验需要的相关环境，同时让我们对linux的环境更加熟悉。

在完成实验一的过程中，我学习并熟悉了很多之前不太熟悉的知识，例如：

* + - 1. Linux的用户管理机制
      2. Linux常用的命令
      3. Linux文件系统结构分化
      4. 环境变量配置的作用
      5. 熟悉了hadoop、spark的配置、使用

电子科技大学信息与软件工程学院

**实 验 报 告**

学 号 2019091604016

姓 名 夏野

（实验） 课程名称 大数据分析与智能计算

实验时间 2021-11-14

实验教师 罗瑜

**电 子 科 技 大 学**

**实 验 报 告**

1. **实验名称：**

MapReduce 与 Spark 内存计算的性能对比

1. **实验学时：**

4学时

1. **实验目的：**
2. 掌握Hadoop的环境搭建；
3. 掌握Hadoop、Linux的基本命令。
4. **实验原理：**
5. MapReduce是一种编程模型，用于大规模数据集（大于1TB）的并行运算。概念"Map（映射）"和"Reduce（归约）"，是它们的主要思想，都是从函数式编程语言里借来的，还有从矢量编程语言里借来的特性。它极大地方便了编程人员在不会分布式并行编程的情况下，将自己的程序运行在[分布式系统](https://baike.baidu.com/item/%E5%88%86%E5%B8%83%E5%BC%8F%E7%B3%BB%E7%BB%9F/4905336" \t "/Users/jackieshi/Documents\\x/_blank)上。 当前的软件实现是指定一个Map（映射）函数，用来把一组键值对映射成一组新的键值对，指定并发的Reduce（归约）函数，用来保证所有映射的键值对中的每一个共享相同的键组。
6. Spark是基于内存计算的大数据并行计算框架.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部署在大量的廉价硬件之上,形成集群。Spark是MapReduce的替代方案，而且兼容HDFS、Hive等分布式存储层，可融入Hadoop的生态系统，以弥补缺失MapReduce的不足。
7. **实验内容：**
8. 使用 Hadoop MapReduce、Spark 框架分别运行 wordcount 分析程序；
9. 对比分析 MapReduce 和 Spark 的性能。
10. **实验器材（设备、元器件）：**
11. 软件环境：Ubuntu 20.04、jdk-8u181-linux-x64.tar、Hadoop 2.7.3、spark-3.1.2-bin-hadoop2.7.tgz、Parallels Desktop
12. 系统环境：MacOS
13. **实验步骤：**
14. 创建用户；
15. 安装ssh；
16. 安装Java环境；
17. 安装Hadoop；
18. 测试。
19. **实验结果与分析（含重要数据结果分析或核心代码流程分析）**
20. 将本次实验的数据文件上传到 HDFS 文件系统
21. 建立data目录

在 /hadoop 目录下创建 data 目录，并将准备好的 word.txt 复制到目录中。

|  |
| --- |
| $ mkdir /hadoop/data  $ cp '/home/parallels/Desktop/Parallels Shared Folders/Home/Desktop/word.txt' /hadoop/data |

1. 查看数据集的大小以及数据集的字符串数量

使用如下命令查看 word.txt 文件的大小和字符数，结果如图 1.1 所示。

|  |
| --- |
| $ du -h word.txt  $ wc -c word.txt |

|  |
| --- |
| Text  Description automatically generated  图 1.1 查看数据集大小和字符数 |

使用 more 命令查看数据集的内容，结果如图 1.2 所示。

|  |
| --- |
| Text  Description automatically generated  图 1.2 查看数据集内容 |

1. 将文件上传到HDFS/wordcount目录中

在分布式文件系统中创建 /wordcount 目录，并将用于统计的 word.txt 文件上传到 HDFS/wordcount 目录下。

使用 -ls 命令查看文件是否上传成功，结果如图 1.3 所示。

|  |
| --- |
| Text  Description automatically generated  图 1.3 上传文件 |

1. MapReduce 实现 WordCount 实例
2. 创建/Hadoop/data/mapreduce，并进入到/Hadoop/data/mapreduce目录下，运行代码如下所示

Text

Description automatically generated

1. 编写 MapReduce WordCount 程序
   1. 编写 Map 端代码

使用vi count\_mapper.py命令创建 count\_mapper.py 文件，编写代码，结果如图 1.4 所示。

|  |
| --- |
| Text  Description automatically generated  图 1.4 Map 端代码 |

* 1. 编写 Reduce 端代码

使用 vi count\_reducer.py命令创建 count\_reducer.py 文件，编写代码，结果如图 1.5 所示。

|  |
| --- |
| Text  Description automatically generated  图 1.5 Reduce 端代码 |

A screenshot of a computer

Description automatically generated with medium confidence生成文件如图1.6所示：

|  |
| --- |
| 图 1.6 mapreduce目录下文件 |

1. 本地测试

完成代码的编写后，使用如下命令在本地进行测试，结果如图 1.7 所示。

|  |
| --- |
| $ head -20 /hadoop/data/word.txt | python3 count\_mapper.py | sort | python3 count\_reducer.py |

|  |
| --- |
| Text  Description automatically generated  图 1.7 本地测试结果 |

1. 运行实例

当我们在本地测试没有问题后，我们使用如下命令在 Hadoop 上运行实例，运行实例结果如图 1.8 所示。

|  |
| --- |
| $ hadoop jar /hadoop/hadoop-3.1.4/share/hadoop/tools/lib/hadoop-streaming-3.1.4.jar -file count\_mapper.py -mapper count\_mapper.py -file count\_reducer.py -reducer count\_reducer.py -input /wordcount/word.txt -output /wordcount-out/mapreduce-out |

|  |
| --- |
| Text  Description automatically generated with medium confidence  图 1.8 运行实例结果 |

1. 查看结果

查看输出目录中的输出文件 part-00000，结果如图 1.9 所示。

|  |
| --- |
| A screen shot of a computer  Description automatically generated with low confidence  图 1.9 查看输出文件 |

1. Spark 实现 WordCount 实例
2. Text

   Description automatically generated创建spark目录
3. 编写 Spark WordCount 程序

创建 /hadoop/data/spark-data 目录，在该目录下使用 vi wordcount.py命令创建 wordcount.py 文件，如图 3.1 所示。

|  |
| --- |
| Text  Description automatically generated  图 3.1 wordcount.py 程序 |

1. 运行实例

使用如下命令运行实例，结果如图 3.2 所示。

|  |
| --- |
| $ /hadoop/app/spark/bin/spark-submit --master spark://localhost:7077 /hadoop/data/spark-data/wordcount.py |

|  |
| --- |
| Text  Description automatically generated  图 3.2 运行实例结果 |

1. 查看结果

运行完成后，我们同样使用如下命令查看输出文件 part-00000，结果如图 3.3 所示。

|  |
| --- |
| $ hadoop fs -tail /wordcount-out/spark-out1/part-00000 |

|  |
| --- |
| Text  Description automatically generated图 3.3 查看输出文件 |

1. 实验结果
2. Mapreduce处理wordcount：作业的执行过程主要分为Map阶段、中间结果排序与传递阶段和Reduce阶段。其具体运行过程大致为：
   1. 首先将数据分块, 并转换为键值对形式;
   2. 每个Map任务根据*key*值对数据进行分区, 即, 同一个分区的数据将会被发送到同一个Reduce节点上;
   3. 对每一个分区进行排序, 同时将可以合并的数据进行合并(如两个(A, 1)将会被合并成(A, 2))
   4. 每一个Reduce从各个Map节点通过网络传输复制相应的数据;
   5. 将不同Map节点的数据进行归并.
3. spark处理wordcount：
   1. 第1次遍历:生成RDD及记录依赖关系
      1. 从外部文件中读取数据生成第1个RDD(ParallelCollectionRDD), 分区数为3;
      2. 将每一个分区再进行哈希分块, 分块数对应阶段2中的任务(task)数目, 每一个分块单独形成一个文件进行数据传输;
      3. 将中间数据文件进行shuffle形成ShuffledRDD, 并在ShuffledRDD中记录下其父RDD以及依赖关系;
      4. 通过对ShuffledRDD进行转换操作生成MapPartitionsRDD, 并在MapPartitionsRDD中记录下其父RDD以及依赖关系;
      5. 因为在MapPartitionsRDD后为行动操作, 则本作业中所有RDD生成完毕, 第1次遍历结束.
   2. 第2次遍历:划分阶段与任务
      1. 根据第1次遍历每一个RDD中记录的父RDD信息及依赖关系, 从MapPartitionsRDD开始向前进行第2次遍历;
      2. 首先进行阶段划分, 若向前遍历时遇到窄依赖关系, 则将依赖的RDD加入到当前阶段; 若遇到宽依赖关系, 则形成一个新的阶段继续向前遍历.在本例中, ParallelCollectionRDD与ShuffledRDD为宽依赖关系, ShuffledRDD与MapPartitionsRDD为窄依赖关系, 所以会生成两个阶段, 阶段1中包括Parallel CollectionRDD, 阶段2中包括ShuffledRDD和MapPartitionsRDD;
      3. 在每一个阶段内部进行任务划分, 任务数目为阶段内部RDD的分区数, 在阶段1中有3个任务, 阶段2中有2个任务;
   3. 第3次遍历:执行任务根据第2次遍历划分的阶段与任务进行作业的具体执行, 首先执行阶段1中的任务, 执行完毕后再执行阶段2中的任务, 输出数据.