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.