Hadoop和Spark性能比较

实验二

信息与软件工程学院

大数据分析与智能计算

实验内容

- Hadoop MapReduce 运行wordcount程序
- Spark运行wordcount程序
- 对比评估两者的运行时间

推荐实验环境

- ubuntu-18.04及以上环境（实验指导书示例使用20.04环境）
- jdk-8u261-linux-x64.tar.gz
- Hadoop 3.1.4
- spark-3.0.1-bin-hadoop3.2.tgz

Hadoop MapReduce概述

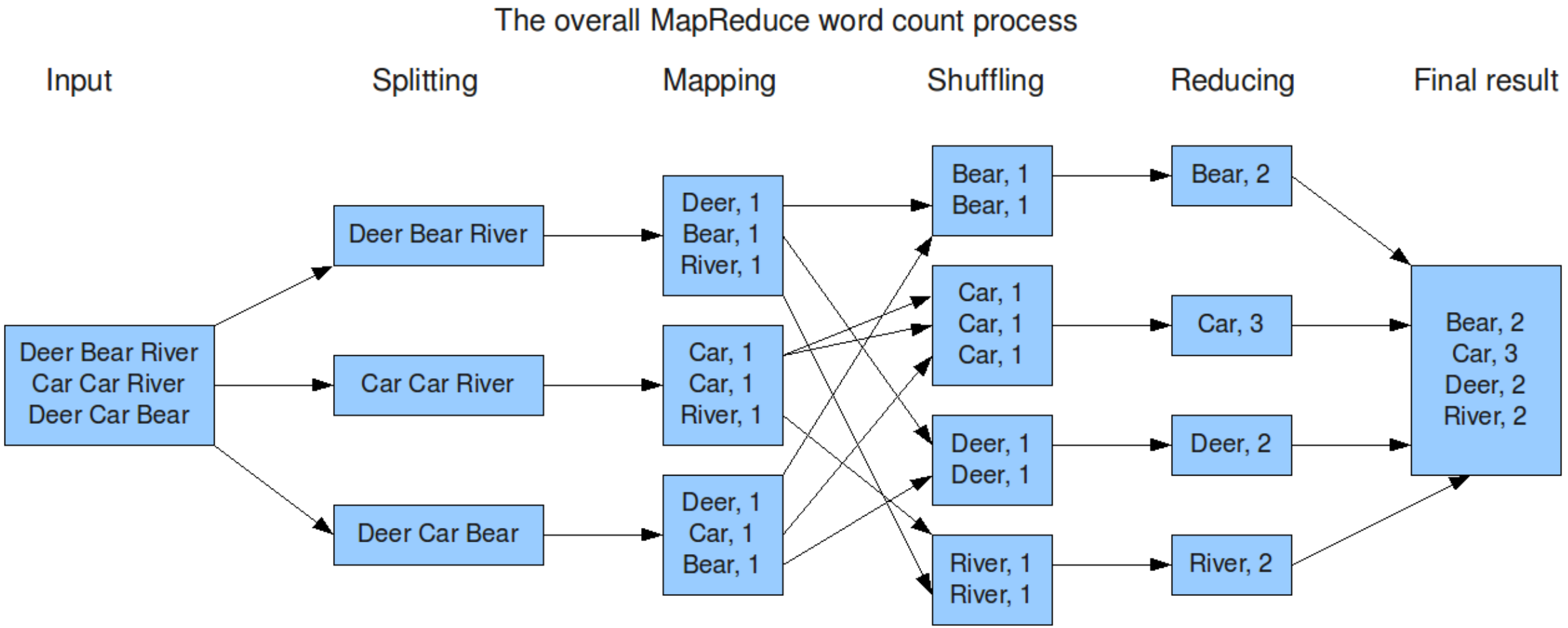
Hadoop 的设计思路来源于 Google 的 GFS 和 MapReduce。它是一个开源软件框架，通过在集群计算机中使用简单的编程模型，可编写和运行分布式应用程序处理大规模数据。一个完整的MapReduce程序在Yarn中执行过程如下：

- （1）ResourcManager JobClient向ResourcManager提交一个job。
- （2）ResourcManager向Scheduler请求一个供MRAppMaster运行的container，然后启动它。
- （3）MRAppMaster启动起来后向ResourcManager注册。
- （4）ResourcManagerJobClient向ResourcManager获取到MRAppMaster相关的信息，然后直接与MRAppMaster进行通信。

Hadoop MapReduce概述

- （5）MRAppMaster算splits并为所有的map构造资源请求。
- （6）MRAppMaster做一些必要的MR OutputCommitter的准备工作。
- （7）MRAppMaster向RM(Scheduler)发起资源请求，得到一组供map/reduce task运行的container，然后与NodeManager一起对每一个container执行一些必要的任务，包括资源本地化等。
- （8）MRAppMaster 监视运行着的task 直到完成，当task失败时，申请新的container运行失败的task。
- （9）当每个map/reduce task完成后，MRAppMaster运行MR OutputCommitter的cleanup 代码，也就是进行一些收尾工作。
- （10）当所有的map/reduce完成后，MRAppMaster运行OutputCommitter的必要的job commit或者abort APIs。
- （11）MRAppMaster退出。

Hadoop MapReduce概述



Spark概述

- Spark 是一个基于内存计算的开源的集群计算系统，目的是让数据分析更加快速。
- Spark 提供了基于内存的计算集群，在分析数据时将数据导入内存以实现快速查询，“速度比”基于磁盘的系统，如比 Hadoop 快很多。
- Spark 最初是为了处理迭代算法，如机器学习、图挖掘算法等，以及交互式数据挖掘算法而开发的。在这两种场景下，Spark 的运行速度可以达到 Hadoop 的几百倍。

数据集准备

- 请自行准备本实验使用的数据集，保存成word.txt文件，上传到实验环境。
- word.txt请使用大于500M的文本
- （1）查看数据集的大小：du -h word.txt
- （2）查看数据集的字符串数：wc -w word.txt
- （3）查看字符集的内容：more word.txt

实验步骤

- **一、实验前准备**

$ su - hadoop

口令输入：hadoop

$ bash

$ echo $HADOOP_HOME

/hadoop/hadoop

上述输出确认hadoop的环境变量设置有效，如果无效则激活环境变量：

$ source ~/.bash_profile

一、实验前准备

启动ssh,口令输入：hadoop

- hadoop@357987c120a9:~$ sudo service ssh start
- [sudo] password for hadoop:
- [ok] Starting OpenBSD Secure Shell server: sshd.
- hadoop@357987c120a9:~$

二. 启动Hadoop

启动命令为：$ start-all.sh

检查是否运行成功#执行jps命令可以查看到hadoop的几个主要进程：

$ jps

```
3409 NodeManager
3733 Jps
2698 NameNode
3258 ResourceManager
2843 DataNode
3084 SecondaryNameNode
```

三.启动spark

①首先启动master

```
hadoop@ubuntu:~/app/spark/sbin$ ./start-master.sh
starting org.apache.spark.deploy.master.Master, logging to /hadoop/app/spark/lo
gs/spark-hadoop-org.apache.spark.deploy.master.Master-1-ubuntu.out
```

- $ cd /hadoop/app/spark/sbin/
- $ ./start-master.sh

②启动slave

```
hadoop@ubuntu:~/app/spark/sbin$ ./start-slave.sh spark://127.0.0.1:7077
starting org.apache.spark.deploy.worker.Worker, logging to /hadoop/app/spark/lo
gs/spark-hadoop-org.apache.spark.deploy.worker.Worker-1-ubuntu.out
```

$ ./start-slave.sh spark://127.0.0.1:7077

查看Master和Worker进程是否启动

- $ jps

```
hadoop@ubuntu:~/app/spark/sbin$ jps
11040 ResourceManager
10866 SecondaryNameNode
11203 NodeManager
10471 NameNode
11946 Master
11994 Jps
10667 DataNode
```

四.将本次实验的数据文件上传到HDFS文件系统

可以建立一个/hadoop/data目录。

- $ mkdir /hadoop/data
- $ cd /hadoop/data

将word.txt上传至该目录下。并查看该目录下是否有了word.txt文件

- $ ls -l

查看数据集的大小：

- $ du -h word.txt

查看数据集的字符串数：

- $ wc -c word.txt

```
hadoop@ubuntu:~$ ls -l word.txt
-rw-rw-r-- 1 hadoop hadoop 770852111 Nov 10 17:16 word.txt
hadoop@ubuntu:~$ du -h word.txt
736M    word.txt
hadoop@ubuntu:~$ wc -c word.txt
770852111 word.txt
```

实验步骤

查看字符集的内容：

- $ more word.txt

more命令说明： more命令会以一页一页的显示方便使用者逐页阅读，而最基本的指令就是按空白键（space）就往下一页显示，按 b 键就会往回（back）一页显示，而且还有搜寻字串的功能 。more命令从前向后读取文件，因此在启动时就加载整个文件。

- 空格键 向下滚动一屏
- q 退出more

实验步骤

- **四.将本次实验的数据文件上传到HDFS文件系统**

将文件上传到HDFS/wordcount

- $ hadoop fs -mkdir /wordcount
- $ hadoop fs -put /hadoop/word.txt /wordcount
- $ hadoop fs -ls -R /wordcount

```
hadoop@ubuntu:~$ hadoop fs -put /hadoop/word.txt /wordcount
hadoop@ubuntu:~$ hadoop fs -ls -R /wordcount
-rw-r--r--   1 hadoop supergroup  770852111 2020-11-10 01:33 /wordcount/word.txt
```

五.MapReduce实现WordCount实例（Python）

创建/hadoop/data/mapreduce，并进入到/hadoop/data/mapreduce目录下

- $ mkdir /hadoop/hadoop-data
- $ cd /hadoop/hadoop-data

第一步：在这个目录下首先编写MapReduce WordCount 代码。

实验步骤

①首先编写map阶段的代码，创建一个Python程序，命名为**count_mapper.py**

$ vi count_mapper.py

```
#!/usr/bin/env python3
import sys
for line in sys.stdin:
    line = line.strip()
    words = line.split()
    for word in words:
        print("%s\t%s" % (word, 1))
```

- 脚本语言的第一行，目的就是指出，这个文件中的代码用什么可执行程序去运行（设备上可能安装了多个版本的Python）。
- #!是特殊的表示符，其后面跟的是此解释此脚本的解释器的路径。是告诉操作系统执行这个脚本的时候，调用/usr/bin/env下的python3解释器

实验步骤

②编写Reduce阶段的代码，创建Python程序，命名为count_reducer.py

- **$ vi count_reducer.py**

```
#!/usr/bin/env python3
from operator import itemgetter
import sys
current_word = None
current_count = 0
word = None
for line in sys.stdin:
    line = line.strip()
    word, count = line.split('\t', 1)
    try:
        count = int(count)
    except ValueError:
        continue
    if current_word == word:
        current_count += count
    else:
        if current_word:
            print ("%s\t%s" % (current_word,
current_count))
        current_count = count
        current_word = word
if current_word == word:
print ("%s\t%s" % (current_word, current_count))
```

实验步骤

第二步：程序编写完成后，首先在本地测试一下map和reduce，命令及图片如下：

- $ head -20 /hadoop/word.txt | python3 count_mapper.py | sort | python3 count_reducer.py
- 如下图，就证明map和reduce程序编写成功。

```
hadoop@ubuntu:~/hadoop-data$ head -20 /hadoop/word.txt | python3 count_mapper.py | so
rt | python3 count_reducer.py
***     2
2020    1
#63695] 1
9,      1
almost  1
and     2
anyone  1
anywhere        1
are     2
at      2
Author: 1
away    1
before  1
by      1
```

实验步骤

第三步：运行该实例，命令如下：

- $ 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

第四步：查看结果

- $ hadoop fs -tail /wordcount-out/mapreduce-out/part-00000

六.Spark实现WordCount实例（python）

创建/hadoop/data/spark，并进入到/hadoop/data/spark目录下

- $ mkdir /hadoop/spark-data
- $ cd /hadoop/spark-data

第一步：首先编写Spark WordCount 代码，创建Python程序，命名为“wordcount.py”。

- $ vi wordcount.py

实验步骤

- wordcount.py

```
#!/usr/bin/env python3
from pyspark import SparkContext
inputFile = 'hdfs://localhost:9000/wordcount/word.txt'
outputFile = 'hdfs://localhost:9000/wordcount-out/spark-out'
sc = SparkContext('local', 'wordcount')
text_file = sc.textFile(inputFile)

counts = text_file.flatMap(lambda line: line.split(' ')).map(lambda word: (word,
1)).reduceByKey(lambda a, b: a+b)

counts.saveAsTextFile(outputFile)
```

实验步骤

第二步：运行该实例

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

第三步：查看运行结果

```
$ hadoop fs  -tail /wordcount-out/spark-out/part-00000
```

实验内容

实验二内容：

- 分析对比使用Hadoop的MapReduce和Spark两者的计算速度。

实验二加分内容：

Word.txt使用中文文本，在hadoop和Spark进行计算之前进行中文分词再做计算。