第 5 章 MapReduce 计算引擎应用

★本章导读★

本章将对 MapReduce 计算引擎的应用做更深一步的介绍。首先，本章节通过介绍 MapReduce 官方例程的实践，使读者更直观的认识基于 MapReduce 计算任务的应用，并对 MapReduce 任务的结果进行分析，做到知因知果。最后，本章会手把手带读者开发自己的第一个基于 Python 的 MapReduce 任务，初步上手体验大数据计算引擎的“快感”。

★知识要点★

通过本章内容的学习，读者将掌握以下知识：

- MapReduce 官方例程的实践
- MapReduce 日志分析
- 基于 Python 的 MapReduce 任务开发流程
- 编写第一个 MapReduce 程序

5.1 你的第一个 MapReduce 程序

MapReduce 计算引擎适用于海量数据离线计算分析。本小节将通过对 Hadoop 自带的官方例程，带读者了解 MapReduce 计算引擎的强大以及广泛应用。

5.1.1 Hadoop 官网实例程序介绍

Hadoop 中自带的 hadoop-mapreduce-examples-2.6.5.jar 中包含一些应用的实例程序，如计算圆周率π（Pi）、单词统计 WordCount 等等。

圆周率π（Pi）的计算这里我们进行简单的介绍和实践。计算圆周率π（Pi）有很多种方法，在 hadoop examples 代码中的注释写的是：采用 Quasi-Monte Carlo 算法来估算π（Pi）的值，该方法叫做蒙特卡洛算法。我们可以通过一个简单的例子带读者理解它的原理：我们取一个单位的正方形（1×1），里面做一个半径为 1 的内切圆（单位圆），我们可以想象往这个图像中投飞镖，则飞镖的落点会出现两种情况：单位圆内，正方形内（单位圆外），则有如下公式：

$$\frac{单位正方形面积}{内切单位圆面积} = \frac{单位正方形内的飞镖数}{内切单位圆内的飞镖数}$$

那么，通过计算飞镖个数就可以把单位圆面积算出来，通过面积，再把圆周率计算出来。不难得知，所计算的π（Pi）的精度与投掷的飞镖的次数成正比。

Hadoop examples 代码中的计算圆周率π（Pi）程序有两个参数：第一个参数为 MapReduce 中 map 的个数；第二个参数为取样个数，即每个 map 投掷的次数。由此可知：

$$总投掷次数 = map\ 个数 \times 每个\ map\ 投掷的次数$$

$$\pi\,(\mathrm{Pi}) = 4 \times \frac{\text{投掷内切单位圆内的次数}}{\text{总投掷次数}}$$

5.1.2 提交集群 MapReduce 程序

本小节将进行 Hadoop 官方 MapReduce 例程的实践。首先，需要启动 Hadoop 集群，保证 Hadoop 集群各个节点处于健康正常运行的状态。执行如下命令，进入到 Hadoop 软件包提供的例程代码目录，找到例程代码，并提交 MapReduce 任务，具体步骤如下。

步骤 01：进入 Hadoop 官方例程代码目录

在 master 节点，输入如下命令。

```
cd  /usr/local/src/hadoop-2.6.5/share/hadoop/mapreduce
ls
```

运行后，终端屏幕上可以看到 Hadoop 提供的一些例程 jar 包，如图 5-1 所示。

```
[root@master mapreduce]# ls
hadoop-mapreduce-client-app-2.6.5.jar        hadoop-mapreduce-client-jobclient-2.6.5-tests.jar
hadoop-mapreduce-client-common-2.6.5.jar     hadoop-mapreduce-client-shuffle-2.6.5.jar
hadoop-mapreduce-client-core-2.6.5.jar       hadoop-mapreduce-examples-2.6.5.jar
hadoop-mapreduce-client-hs-2.6.5.jar         lib
hadoop-mapreduce-client-hs-plugins-2.6.5.jar lib-examples
hadoop-mapreduce-client-jobclient-2.6.5.jar  sources
```

图 5-1 官方例程 jar 包

其中，红色框选中的“hadoop-mapreduce-examples-2.6.5.jar”，即本小节实践使用到的官方例程代码。

步骤 02：进入 Hadoop 官方例程代码目录

在终端中输入如下命令及参数，向 Hadoop 集群提交 MapReduce 任务。

```
hadoop  jar  hadoop-mapreduce-examples-2.6.5.jar  pi  10  15
```

通过终端页面可以看到 MapReduce 任务提交的相关信息，包括 map 个数、每个 map 采样个数、map/reduce 任务进行情况等，如图 5-2 所示。

```
[root@master mapreduce]# hadoop jar hadoop-mapreduce-examples-2.6.5.jar pi 10 15
Number of Maps  = 10
Samples per Map = 15
Wrote input for Map #0
Wrote input for Map #1
Wrote input for Map #2
Wrote input for Map #3
Wrote input for Map #4
Wrote input for Map #5
Wrote input for Map #6
Wrote input for Map #7
Wrote input for Map #8
Wrote input for Map #9
Starting Job
21/10/04 11:26:59 INFO client.RMProxy: Connecting to ResourceManager at master/192.168.75.100:8032
21/10/04 11:27:00 INFO input.FileInputFormat: Total input paths to process : 10
21/10/04 11:27:00 INFO mapreduce.JobSubmitter: number of splits:10
21/10/04 11:27:00 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1630220573843_0001
21/10/04 11:27:01 INFO impl.YarnClientImpl: Submitted application application_1630220573843_0001
21/10/04 11:27:01 INFO mapreduce.Job: The url to track the job: http://master:8088/proxy/applicatio
n_1630220573843_0001/
21/10/04 11:27:01 INFO mapreduce.Job: Running job: job_1630220573843_0001
21/10/04 11:27:14 INFO mapreduce.Job: Job job_1630220573843_0001 running in uber mode : false
21/10/04 11:27:14 INFO mapreduce.Job:  map 0% reduce 0%
```

图 5-2 MapReduce 任务运行日志

5.1.3 查询 MapReduce 任务

在上一步完成了官方例程提供的计算圆周率 π （Pi）的 MapReduce 任务，通过终端可以看到 map/reduce 任务的进度，同样通过 Yarn 页面也可以看到 MapReduce 任务的进展情况，如图 5-3 所示。

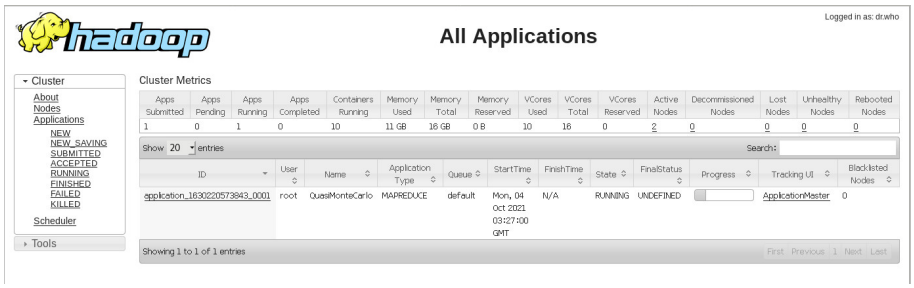

图 5-3 Yarn Application 页面

从图 5-3 中可以看到，该 MapReduce 任务的 State 为“RUNNING”，说明该 MapReduce 任务正在运行中，同样在 Progress 中的进度条可以看到该 MapReduce 任务还未完成。

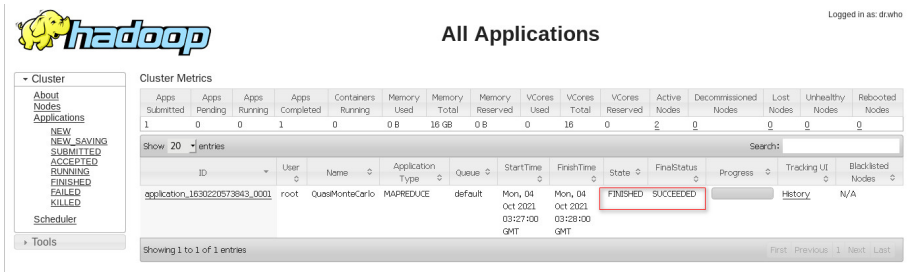

图 5-4 MapReduce 任务运行结束

当任务完成时，刷新 Yarn 页面，如图 5-4 所示，可以看出任务的 State 已经变为“FINISH”，FinalStatus 的结果为“SUCCEEDED”，Progress 进度条已经完成，说明当前 MapReduce 任务已经正常进行并且成功完成。

5.1.4 MapReduce 程序结果分析

上一小节，我们介绍了通过 Yarn 页面查看 MapReduce 任务进度的方法。本小节我们将介绍在任务提交的终端页面中输出展示的相关信息，MapReduce 任务完整输出日志如下。

```
[root@master]# hadoop  jar  hadoop-mapreduce-examples-2.6.5.jar  pi  10  15
Number of Maps  = 10
Samples per Map = 15
Wrote input for Map #0
Wrote input for Map #1
Wrote input for Map #2
Wrote input for Map #3
Wrote input for Map #4
Wrote input for Map #5
```

```
	Wrote input for Map #6
	Wrote input for Map #7
	Wrote input for Map #8
	Wrote input for Map #9
	Starting Job
	22/07/17 21:06:20 INFO client.RMProxy: Connecting to ResourceManager at emr-header-1.cluster-46685/10.73.153.26:8032
	22/07/17 21:06:20 INFO input.FileInputFormat: Total input files to process : 10
	22/07/17 21:06:21 INFO mapreduce.JobSubmitter: number of splits:10
	22/07/17 21:06:21 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1657709793103_1918
	22/07/17 21:06:21 INFO impl.YarnClientImpl: Submitted application application_1657709793103_1918
	22/07/17 21:06:21 INFO mapreduce.Job: The url to track the job: http://emr-header-1.cluster-46685:8088/proxy/application_1657709793103_1918/
	22/07/17 21:06:21 INFO mapreduce.Job: Running job: job_1657709793103_1918
	22/07/17 21:06:26 INFO mapreduce.Job: Job job_1657709793103_1918 running in uber mode : false
	22/07/17 21:06:26 INFO mapreduce.Job:  map 0% reduce 0%
	22/07/17 21:06:32 INFO mapreduce.Job:  map 20% reduce 0%
	22/07/17 21:06:33 INFO mapreduce.Job:  map 80% reduce 0%
	22/07/17 21:06:34 INFO mapreduce.Job:  map 100% reduce 0%
	22/07/17 21:06:36 INFO mapreduce.Job:  map 100% reduce 100%
	22/07/17 21:06:36 INFO mapreduce.Job: Job job_1657709793103_1918 completed successfully
	22/07/17 21:06:36 INFO mapreduce.Job: Counters: 50
		File System Counters
			FILE: Number of bytes read=67
			FILE: Number of bytes written=1897308
			FILE: Number of read operations=0
			FILE: Number of large read operations=0
			FILE: Number of write operations=0
			HDFS: Number of bytes read=2810
			HDFS: Number of bytes written=215
			HDFS: Number of read operations=43
			HDFS: Number of large read operations=0
			HDFS: Number of write operations=3
		Job Counters
			Launched map tasks=10
			Launched reduce tasks=1
			Data-local map tasks=5
			Rack-local map tasks=5
			Total time spent by all maps in occupied slots (ms)=1728519
			Total time spent by all reduces in occupied slots (ms)=184710
			Total time spent by all map tasks (ms)=36777
			Total time spent by all reduce tasks (ms)=1965
			Total vcore-milliseconds taken by all map tasks=36777
			Total vcore-milliseconds taken by all reduce tasks=1965
			Total megabyte-milliseconds taken by all map tasks=55312608
```

```
                Total megabyte-milliseconds taken by all reduce tasks=5910720
        Map-Reduce Framework
                Map input records=10
                Map output records=20
                Map output bytes=180
                Map output materialized bytes=250
                Input split bytes=1630
                Combine input records=0
                Combine output records=0
                Reduce input groups=2
                Reduce shuffle bytes=250
                Reduce input records=20
                Reduce output records=0
                Spilled Records=40
                Shuffled Maps =10
                Failed Shuffles=0
                Merged Map outputs=10
                GC time elapsed (ms)=1040
                CPU time spent (ms)=5790
                Physical memory (bytes) snapshot=4843855872
                Virtual memory (bytes) snapshot=36820439040
                Total committed heap usage (bytes)=7128743936
        Shuffle Errors
                BAD_ID=0
                CONNECTION=0
                IO_ERROR=0
                WRONG_LENGTH=0
                WRONG_MAP=0
                WRONG_REDUCE=0
        File Input Format Counters
                Bytes Read=1180
        File Output Format Counters
                Bytes Written=97
Job Finished in 16.187 seconds
Estimated value of Pi is 3.17333333333333333333
```

可以从最后一行日志中看出 MapReduce 任务的输出结果，如图 5-5 所示。

```
Job Finished in 63.254 seconds
Estimated value of Pi is 3.17333333333333333333
[root@master mapreduce]#
```

图 5-5 圆周率π（Pi）估算结果

当 MapReduce 任务运行结束之后，我们通过终端输出的末尾信息可以看到该任务总共耗时 63.254 秒，以及任务输出的结果，值得注意的是，hadoop example 中圆周率π（Pi）并不是计算得到的精准值，而是对圆周率的估算值。根据上节的介绍，读者通过调整该 MapReduce 任务的两个参数，即 map 个数和每个 map 的采样个数，输出的结果也会有所改变，读者可以自行尝试。

批注 [雷1]: 上页空白很大，建议直接将代码复制成正文，然后加上浅灰色的底纹

批注 [HZ2R1]: OK

5.2 手把手 WordCount 实战

5.2.1 WordCount 介绍

本小节将详细介绍 MapReduce 中的“HelloWord”入门实践——WordCount，即单词统计，这也是最能体现 MapReduce 思想的程序之一。WordCount 的功能是统计文本文件中每个单词出现的次数。

WordCount 的 MapReduce 处理过程包括 4 个阶段，具体过程如下所示。

步骤 01：Split

Split 过程会将文件进行分割，并将文件按行分割形成<key, value>对，如图 5-6 所示，这一步骤是由 MapReduce 框架自动完成，其中偏移量(即 key 值)包括了回车所占的字符数（注意：在 Windows 和 Linux 环境中不同）。

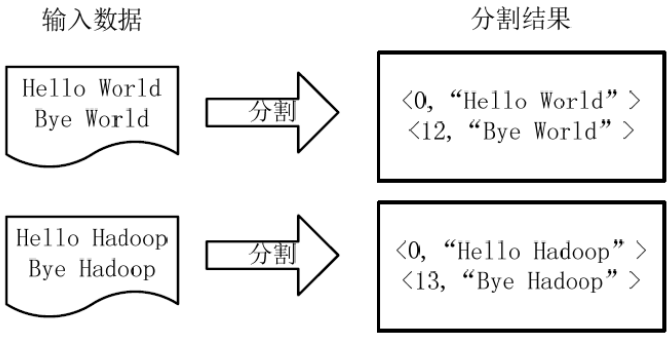

图 5-6 Split 阶段

步骤 02：Mapper

MapReduce 框架通过用户定义的 map 方法将 split 片段进行处理，把分割好的<key, value>重新处理，生成新的<key, value>对，如下图 5-7 所示。

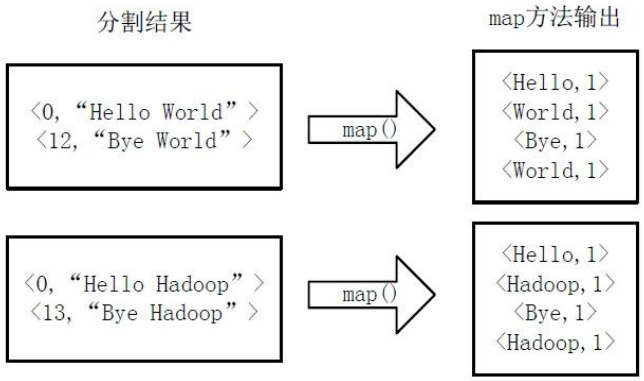

图 5-7 Mapper 阶段

步骤 03：Combine

map 方法输出新的<key, value>之后，Mapper 端会根据 key 值对新生成的<key, value>进行排序，并执行 Combine 过程，将 key 值相同的 value 值累加，相当于对 map 的输出进行了一次预聚合操作，这样也可以减少 Reducer 端的计算量，这一步得到 Mapper 的最终输出结果，如下图 5-8 所示。

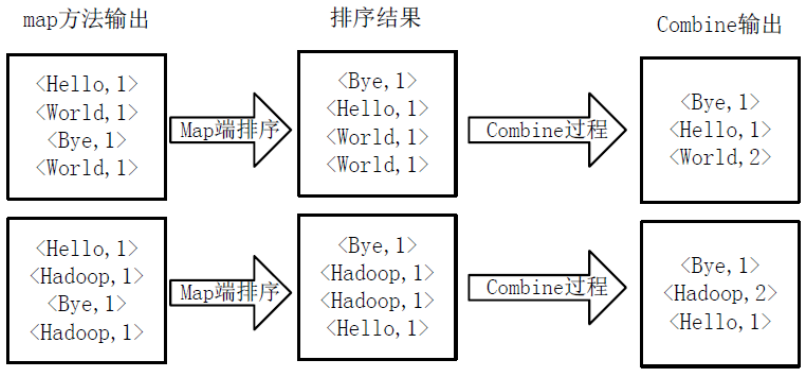

图 5-8 Combine 过程

步骤 04：Reducer

Reducer 先对 Mapper 接收的数据进行排序，再交由用户自定义的 reduce 方法进行处理，得到新的<key, value>对，并作为 WordCount 的输出结果，如图 5-9 所示。

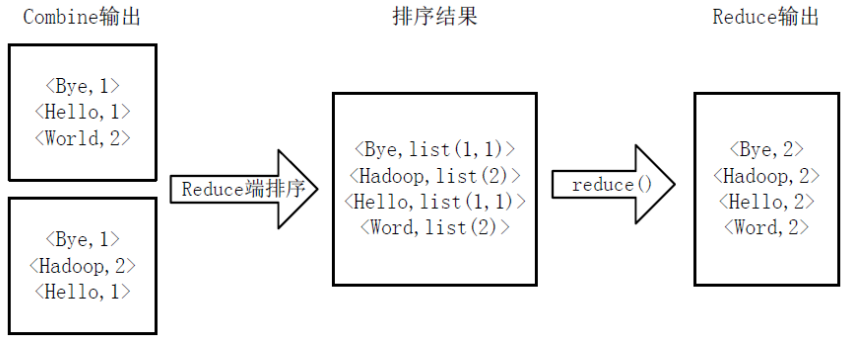

图 5-9 Reducer 阶段

通过上述 4 个步骤就完成了基于 MapReduce 计算框架的 WordCount 单词统计任务。其中最核心的部分即为 Mapper 阶段和 Reducer 阶段，这两个阶段分别完成了对单词的映射变换和单词的规约统计。

根据上述步骤我们就可以完成 MapReduce 程序的开发工作。在下面的小节中将详细介绍 mapper.py 和 reducer.py 的代码开发和基于 Python 的 MapReduce 程序如何提交 Hadoop 集群运行。

批注 [雷3]: 本段没有承上的表述，需要对上面步骤内容进行总结

5.2.2 带你编写 mapper.py

在开始本小节 mapper.py 的开发之前，我们首先要了解数据结构和内容。本书选用了《The DaVinci Code》这本书的文档作为 WordCount 实验的数据，文档的简要内容如下图 5-10 所示。

```
TheDaVinciCode - FOR BLYTHE... AGAIN. MORE THAN EVER

FOR BLYTHE... AGAIN. MORE THAN EVER.

Acknowledgments
First and foremost, to my friend and editor, Jason Kaufman, for working so hard
on this project and for truly understanding what this book is all about. And to
the incomparable Heide Lange-tireless champion of The Da Vinci Code, agent
extraordinaire, and trusted friend.
I cannot fully express my gratitude to the exceptional team at Doubleday, for
their generosity, faith, and superb guidance. Thank you especially to Bill
Thomas and Steve Rubin, who believed in this book from the start. My thanks
also to the initial core of early in-house supporters, headed by Michael
Palgon, Suzanne Herz, Janelle Moburg, Jackie Everly, and Adrienne Sparks, as
well as to the talented people of Doubleday's sales force.
```

图 5-10 《The DaVinci Code》数据展示

接下来，在 master 节点创建本章相关代码文件夹，执行如下命令。

```
cd /usr/code
mkdir Chapter_05
```

将《The DaVinci Code》小说文档上传至“Chapter_05”目录下。

由图 5-10 中所示文档内容可知，英文单词之间是通过空格来区分的（这里不考虑标点符号，读者可以进行尝试），而 mapper 的任务是将数据拆分成<key, value>对，所以，根据任务需求，在“Chapter_05”文件夹下创建 mapper.py 文件，mapper.py 的实现代码如下所示。

```
# -*- coding:utf-8 -*-
import sys

for line in sys.stdin:
    line = line.strip()
    words = line.split()
    for word in words:
        print("{}\t{}".format(word.strip(), 1))
```

在编写完成 mapper.py 代码之后，通过如下命令对 mapper.py 逻辑功能进行测试验证。

```
echo  " ha py map py re py had "|  python  mapper.py
```

运行结果结果如图 5-11 所示。

```
[root@master Chapter_05]# echo "ha py map py re py had" | python mapper.py
ha      1
py      1
map     1
py      1
re      1
py      1
had     1
```

图 5-11 mapper.py 本地管道执行结果

从图 5-11 中可以看出，我们利用 mapper.py 代码对输入的“ha py map py re py had”中的每个单词映射成了“word 1”结构的键值对，其中“word”指“ha py map py re py had”中的单词。

> 温馨提示：
> "|" 起管道的作用，将前方的内容传递给后面命令。

5.2.3 带你编写 reducer.py

reducer.py 阶段是对 map 阶段生成的键值映射对组合生成每个词的计数。reducer.py 实现代码如下所示。

```
# -*- coding:utf-8 -*-
import sys

current_word = None
current_sum = 0
word = None

for line in sys.stdin:
    word, count = line.strip().split('\t', 1)
    try:
        count = int(count)
    except ValueError:
        continue
    if current_word == word:
        current_sum += count
    else:
        if current_word:
            print("{}\t{}".format(current_word, current_sum))
        current_sum = count
        current_word = word

if word == current_word:
    print("{}\t{}".format(current_word, current_sum))
```

同样，在编写完成 reducer.py 代码之后，通过如下命令对 reducer.py 逻辑功能进行测试验证。

```
echo "ha py map py re py had" | python mapper.py | sort -k1,1 | python reducer.py
```

结果如图 5-12 所示。

```
[root@master Chapter_05]# echo "ha py map py re py had" | python mapper.py | sort -k1,1 | python reducer.py
ha      1
had     1
map     1
py      3
re      1
```

图 5-12 reducer.py 本地管道执行结果

> 温馨提示：
>
> 其中"sort -k1,1"模拟 MapReduce 中的排序部分。
>
> k1,1 ：从第 1 个字段开始到第 1 个字段结束，作为 key，就是指第一个字段。

5.2.4 提交 WordCount 程序

在编写完成 mapper.py、reducer.py 并进行验证之后，接下来需要将代码提交到 Hadoop 集群，利用 MapReduce 计算引擎实现 WordCount 的功能。

本书 Hadoop MapReduce 是基于 Python 语言实现的，所以需要通过 Hadoop Streaming 的方式实现提交代码的任务。Hadoop Streaming 是 Hadoop 提供的一种编程工具，允许用户用任何可执行程序和脚本作为 mapper 和 reducer 来完成 Map/Reduce 任务，开发者可以使用 Python、Ruby、Golang 等任何熟悉的语言来完成我们大数据探索的需求。

Hadoop Streaming 有如下特点：

1. 开发效率高

（1） 方便移植 Hadoop 平台，只需按照一定的格式从标准输入读取数据、向标准输出写数据就可以。

（2） 原有的单机程序稍加改动就可以在 Hadoop 平台上进行分布式处理。

（3） 容易单机调试。

2. 程序运行效率高

对于 CPU 密集的计算，有些语言如 C/C++编写的程序可能比用 Java 编写的程序效率更高一些。

3. 便于平台进行资源控制

Streaming 框架中通过 limit 等方式可以灵活地限制应用程序使用的内存等资源。

具体步骤如下。

步骤 01：启动 Hadoop 集群

在 master 节点启动 Hadoop 集群，命令如下。

```
cd /usr/local/src/hadoop-2.6.5/sbin
sh ./start-all.sh
```

步骤 02：上传数据到 hdfs

执行代码如下所示。

```
cd /usr/code/ Chapter_05/
hdfs dfs -put The_DaVinci_Code.txt /The_DaVinci_Code.txt
```

通过 HDFS UI 页面可以查看上传到 HDFS 上的数据文件，如图 5-13 所示。

图 5-13 查看 HDFS 上传文件

步骤 03：提交任务代码

执行如下命令，创建提交任务可执行脚本文件。

```
cd /usr/code/ Chapter_05/
touch run.sh
chmod +x run.sh
```

脚本文件 run.sh 代码内容如下所示。

```
STREAM_JAR_PATH=/usr/local/src/hadoop-2.6.5/share/hadoop/tools/lib/hadoop-streaming-2.6.5.jar

hadoop jar $STREAM_JAR_PATH \
-input /The_DaVinci_Code.txt \
```

```
-output /wordcount_out \
-mapper "python mapper.py" \
-reducer "python reducer.py" \
-jobconf "mapred.reduce.tasks=2" \
-file ./mapper.py \
-file ./reducer.py
```

run.sh 脚本文件中第一行是指明用到的 streaming 包的位置，第二行指明原文件在 HDFS 上的路径，第三行是输出结果在 HDFS 上的路径，输出路径原来不能存在，已存在的话会报错，最后两行指明 Map 方法和 Reduce 方法程序路径。

执行以下脚本文件 run.sh 提交任务。

```
sh run.sh
```

提交任务成功后，MapReduce 执行日志如下图 5-14 所示。

```
[root@master Chapter_05]# sh run.sh
21/11/09 22:38:48 WARN streaming.StreamJob: -file option is deprecated, please use generic option -files instead.
21/11/09 22:38:48 WARN streaming.StreamJob: -jobconf option is deprecated, please use -D instead.
21/11/09 22:38:48 INFO Configuration.deprecation: mapred.reduce.tasks is deprecated. Instead, use mapreduce.job.reduces
packageJobJar: [./mapper.py, ./reducer.py, /tmp/hadoop-unjar1669647822021170403/] [] /tmp/streamjob1798495063143362992.jar tmpDir=null
21/11/09 22:38:49 INFO client.RMProxy: Connecting to ResourceManager at master/192.168.75.100:8032
21/11/09 22:38:49 INFO client.RMProxy: Connecting to ResourceManager at master/192.168.75.100:8032
21/11/09 22:38:50 INFO mapred.FileInputFormat: Total input paths to process : 1
21/11/09 22:38:50 INFO mapreduce.JobSubmitter: number of splits:2
21/11/09 22:38:50 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1636466366666_0007
21/11/09 22:38:50 INFO impl.YarnClientImpl: Submitted application application_1636466366666_0007
21/11/09 22:38:50 INFO mapreduce.Job: The url to track the job: http://master:8088/proxy/application_1636466366666_0007/
21/11/09 22:38:50 INFO mapreduce.Job: Running job: job_1636466366666_0007
21/11/09 22:38:56 INFO mapreduce.Job: Job job_1636466366666_0007 running in uber mode : false
21/11/09 22:38:56 INFO mapreduce.Job:  map 0% reduce 0%
21/11/09 22:39:04 INFO mapreduce.Job:  map 50% reduce 0%
21/11/09 22:39:05 INFO mapreduce.Job:  map 100% reduce 0%
21/11/09 22:39:12 INFO mapreduce.Job:  map 100% reduce 100%
21/11/09 22:39:12 INFO mapreduce.Job: Job job_1636466366666_0007 completed successfully
21/11/09 22:39:12 INFO mapreduce.Job: Counters: 50
```

图 5-14 MapReduce 任务执行日志

5.2.5 WordCount 程序结果分析

上一小节中通过执行 run.sh 脚本，将 mapper 和 reducer 的处理程序通过 Hadoop Streaming 提交到了 MapReduce 进行处理统计《The DaVinci Code》小说单词个数。在 run.sh 脚本中指定了 reduce 任务输出目录（HDFS 路径）为“/wordcount_out”，通过 HDFS UI 可以查看是否有输出结果，如下图 5-15 所示。

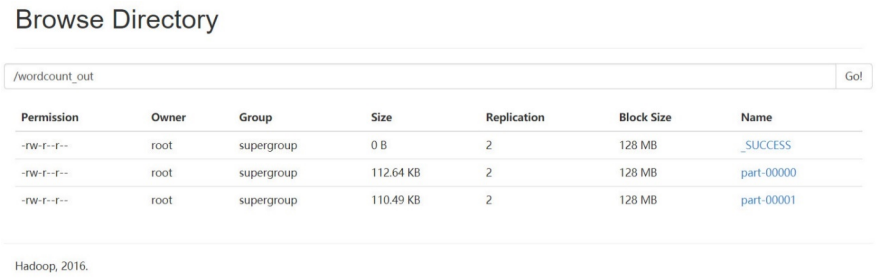

图 5-15 查看 HDFS 输出结果文件

执行成功后，在 HDFS 的“/wordcount_out”目录下可以看到下面两个输出结果的文件，part-00000 和 part-00001，观察两个文件的 Size 可知两个文件的大小都小于 128MB，这是因为我们在

run.sh 脚本中通过-jobconf 参数设置了 reduce task 的个数的原因。

通过如下命令可以查看 HDFS 上生成的数据。

```
hdfs dfs -cat /wordcount_out/part-00000 | head
```

结果如图 5-16 所示。

```
[root@master Chapter_05]# hdfs dfs -cat /wordcount_out/part-00000 | head
"-is    1
"...And 1
"...qui 1
"A      56
"A.     1
"About  2
"Abracadabra?"  1
"Actually       1
"Actually,      9
"Actually,"     3
```

图 5-16 查看 MapReduce 输出结果

如图 5-16 所示，使用 head 命令展示了 wordcount 统计计算结果的前 10 行。

温馨提示：

-jobconf：提交作业的常见配置属性如下：

（1） mapred.map.tasks # map task 数目

（2） mapred.reduce.tasks # reduce task 数目

（3） mapred.text.key.comparator.options # 设置 key 中需要比较的字段或字节范围

（4） num.key.fields.for.partition # 通过参数去选择哪一部分去做 partition

★回顾思考★

01 MapReduce 在实现 WordCount 时主要包含哪些阶段？

答：包括如下阶段：

（1） Split 阶段；

（2） Mapper 阶段；

（3） Combine 阶段；

（4） Reducer 阶段；

02 Hadoop Streaming 提交脚本参数至少包括哪几项？

答：包括如下：

（1） -input <输入数据路径>；

（2） -output <输出数据路径>；

（3） -mapper “python map 代码”
（4） -reducer “python reduce 代码”；
（5） -file <map 代码>；
（6） -file <reduce 代码>；

★练习★

一、选择题

1. MapReduce 不包括哪个阶段（ ）
   A. map
   B. reduce
   C. count
   D. split
2. 哪项是 Hadoop Streaming 脚本参数-jobconf 内容（ ）
   A. mapred.reduce.split
   B. mapred.reduce.count
   C. mapred.map.split
   D. mapred.map.tasks

二、判断题

1. Hadoop Streaming 只限使用 Python/Go 语言进行开发。（×）
2. 蒙特卡洛算法可以用于计算圆周率估值。（√）

三、实战练习

1. 使用 Hadoop 官方例程 jar 包实现圆周率计算。
2. 开发编写 WordCount 代码，并提交集群运行。
3. 开发编写 MapReduce 代码，通过本地管道运行，实现如下内容的统计：

数据：data.txt

```
id_a, id_b, id_c, id_d
id_a, id_a, id_f
id_b, id_b, id_d, id_f, id_a
id_m, id_n
```

需求：统计每个词组“_”后一个字母的个数

本章小结

本章首先通过 MapReduce 开发实战向读者介绍了 Hadoop 官方例程包的使用方法，简要介绍了圆周率 π（Pi）的估算方法和蒙特卡洛算法在 MapReduce 上的应用实例。

本书的核心在基于 Python 语言进行大数据处理和数据分析，所以本章节也手把手带读者从零开始编写了基于 Python 语言的 MapReduce 程序——WordCount 实战。其中，详细介绍了 Hadoop Streaming 平台的优势，最后通过对 Mapper 和 Reducer 代码的本地验证和集群验证方式实现了 WordCount 任务实

战。通过本章节的学习，读者可以亲自感受到大数据处理的便捷之处和大数据处理的思想精髓，为后续章节的学习做铺垫。