实验二：Hadoop 并行编程

一、 实验目的

- 学习 Hadoop 分布式文件系统、MapReduce 编程模型，理解 MapReduce 的思想。
- 学习使用 Hadoop 基本命令，使用 MapReduce 编写简单的并行行程序。
- 编写 WordCount 程序，对样例数据进行词频统计。

二、 任务与要求

- 了解分布式文件系统原理，以及分布式文件系统与本地文件系统的区别。
- 掌握 Hadoop 分布式文件系统-ls、-copyFromLocal、-copyToLocal 等指令的操作。
- 使用 MapReduce 编程模型编写词频统计程序并运行。
- 设计并实现一个简单的分布式文件系统。

三、 实验原理

3.1. MapReduce 理论简介

3.1.1. MapReduce 编程模型

MapReduce 采用”分而治之"的思想，把对大规模数据集的操作，分发给一个主节点管理下的各个分节点共同完成，然后通过整合各个节点的中间结果，得到最终结果。简单地说，MapReduce 就是"任务的分解与结果的汇总”。

在 Hadoop 中，用于执行 MapReduce 任务的机器角色有两个：一个是 JobTracker;另一个是 TaskTracker，JobTracker 是用于调度工作的，TaskTracker 是 用于执行工作的。一个 Hadoop 集群中只有一台 JobTracker。

在分布式计算中，MapReduce 框架负责处理了并行编程中分布式存储、工作调度、负载均衡、容错均衡、容错处理以及网络通信等复杂问题，把处理过程高度抽象为两个函数：map 和 reduce，map 负责把任务分解成多个任务，reduce 负责把分解后多任务处理的结果汇总起来。

需要注意的是，用 MapReduce 来处理的数据集(或任务)必须具备这样的特点：**待处理的数据集可以分解成许多小的数据集，而且每一个小数据集都可以完全并行地进行处理**。

3.2. MapReduce 的执行过程

MapReduce 运行在大规模集群之上,要完成一个并行计算,还需要任务调度、本地计算、Shuffle(洗牌)过程等一系列环节共同支撑计算的过程。

3.2.1 任务调度与执行

MapReduce 任务由一个 JobTracker 和多个 TaskTracker 两类节点控制完成。JobTracker 主要负责调度和管理 TaskTracker,它通常情况下运行在 Master 节点上。JobTracker 将 Mappers 和 Reducers 分配给空闲的 TaskTracker 后，由 TaskTracker 负责这些任务的并行执行。TaskTracker 必须运行在 DataNode 上，所以 DataNode 既是数据的存储节点也是计算节点。JobTracker 也负责监控任务的运行状况如果某个 TaskTracker 发生故障,JobTracker 就会将其负责的任务分配给其他空闲的 TaskTracker 重新执行。MapReduce 框架的这种设计很适合于集群上任务的调度和执行，当然 JobTracker 的故障将引起整个任务失败，在 Hadoop 以后的发行版本中或许会通过运行多个 JobTracker 解决这个问题。

3.2.2 本地计算

把计算节点和数据节点置于同一台计算机上,MapReduce 框架尽最大的努力保证在那些存储了数据的节点上执行计算任务。这种方式有效地减少了数据在网络中的传输降低了任务对网络带宽的需求避免了使网络带宽成为瓶颈，所以“本地计算”可以说是节约带宽最有效的方式，业界称之为“移动计算比移动数据更经济”。也正是因为如此,split 通常情况下应该小于或等于 HDFS 数据块的大小(默认情况下 64MB),从而保证 split 不会跨越两台计算机存储便于“本地计算”。

3.2.3 Shuffle 过程

MapReduce 会将 Mapper 的输出结果按照 key 值分成 R 份(R 是预先定义的 Reducers 的个数),划分时常使用哈希函数，如 Hash(key) mod R。这样可以保证某一范围内的 key 一定由某个 Reducer 来处理从而简化 Reduce 的过程。

3.2.4 合并 Mapper 输出

正如之前所说，带宽资源非常宝贵，所以 MapReduce 允许在 Shuffle 之前先对结果进行合并(Combine 过程),即将中间结果中有相同 key 值的多组<key, value>对合并成一对。Combine 过程和 Reduce 过程类似，很多情况下可以直接使用 reduce 函数，但 Combine 过程是 Mapper 的一部分，在 map 函数之后执行。Combine 过程通常情况下可以有效地减少中间结果的数量，从而减少数据传输过程中的网络流量。值得注意的是,Hadoop 并不保证其会对一个 Mapper 输出执行多少次 Combine 过程，也就是说，开发人员必须保证不论 Combine 过程执行多少次，得到的结果都是一样的。

3.2.5 读取中间过程

在完成 Combine 和 Shuffle 的过程后,Mapper 的输出结果被直接写到本地磁盘。然后，通知 JobTracker 中间结果文件的位置再由 JobTracker 告知 Reducer 到哪个 DataNode.上去取中间结

果。注意所有的 Mapper 产生的中间结果均按其 key 值用同一个哈希函数划分成 R 份,R 个 Reducer 各自负责一段 key 值区间。每个 Reducer 需要向多个 Mapper 节点取得落在其负责的 key 值区间内的中间结果然后执行 reduce 函数,形成一个最终的结果文件。　需要说明的是,Mapper 的输出结果被直接写到本地磁盘而非 HDFS,因为 Mapper 输出的是中间数据，当任务完成之后就可以直接删除了，如果存储在 HDFS 上,HDFS 的备份机制会造成性能的损失。

3.2.6 任务管道

R 个 Reducer 会产生 R 个结果，很多情况下这 R 个结果并不是所需要的最终结果，而是会将这 R 个结果作为另一个计算任务的输入，并开始另一个 MapReduce 任务。

四、 实验步骤

4.1 任务 1. 掌握 Hadoop DFS 常用指令

在实验一中，我们通过 Linux 指令对本地文件系统进行操作，如使用 ls 查看文件/目录信息、使用 cp 进行文件复制、使用 cat 查看文件内容。在分布式文件系统中，也有一套相似的指令，接下来我们需要掌握一些基本的指令。

首先查看 Hadoop DFS 支持哪些指令

```
2018211009@thumm01:~$ hadoop fs
Usage: hadoop fs [generic options]
      [-cat [-ignoreCrc] <src> ...]
      [-copyFromLocal [-f] [-p] [-l] [-d] [-t <thread count>] <localsrc> ... <dst>]
      [-copyToLocal [-f] [-p] [-ignoreCrc] [-crc] <src> ... <localdst>]
      [-cp [-f] [-p | -p[topax]] [-d] <src> ... <dst>]
      [-head <file>]
      [-help [cmd ...]]
      [-ls [-C] [-d] [-h] [-q] [-R] [-t] [-S] [-r] [-u] [-e] [<path> ...]]
      [-mkdir [-p] <path> ...]
      [-moveFromLocal <localsrc> ... <dst>]
      [-moveToLocal <src> <localdst>]
      [-mv <src> ... <dst>]
      [-rm [-f] [-r|-R] [-skipTrash] [-safely] <src> ...]
      [-rmdir [--ignore-fail-on-non-empty] <dir> ...]
      ......
```

上面是 DFS 中常用的指令，这些指令中有一些我们在本地文件系统中也用过，如 ls、cp、mv、rm、mkdir、cat、head，还有一些指令是 DFS 特有的，例如 **copyFromLocal**、**copyToLocal**、**copyFromLocal**、**copyToLocal**，主要用于 DFS 与本地文件系统的数据交换。

首先使用 ls 指令查看 DFS 中根目录下文件/文件夹的信息

```
2018211009@thumm01:~$ hadoop fs -ls /
Found 2 items
drwxr-xr-x   - root supergroup          0   2019-10-18 16:06 /dsjxtjc
-rw-r--r--   3 root supergroup   13588590 2019-10-18 14:50 /wc_dataset.txt
```

可以看到，现在 DFS 根目录下一共有两项，其中 wc_dataset.txt 是实验一中的数据集，dsjxtjc 是一个文件夹，在这个文件夹下面有每位同学的文件夹，例如某位同学的学号是 201921xxxx，那么他/她对应的文件夹为/dsjxtjc/201921xxxx/。为了保证实验过程中不同用户之间不会产生干扰，每位同学只能在自己的文件夹下进行操作。

首先查看自己的文件下的内容:

```
2018211009@thumm01:~$ hadoop fs -ls /dsjxtjc/2018211009
2018211009@thumm01:~$
```

接下来在本地创建一个 test.txt 文件

```
2018211009@thumm01:~$ touch test.txt
2018211009@thumm01:~$ echo "Hello Hadoop" > test.txt
2018211009@thumm01:~$ cat test.txt
Hello Hadoop
```

将本地文件传输至 DFS 中

```
2018211009@thumm01:~$ hadoop fs -copyFromLocal ./test.txt /dsjxtjc/2018211009/
2018211009@thumm01:~$ hadoop fs -cat /dsjxtjc/2018211009/test.txt
Hello Hadoop
```

可以看到文件以及传输到 DFS 上。

copyFromLocal/copyToLocal 用于本地文件系统与 DFS 之间文件的复制，moveFromLocal/moveToLocal 用于本地文件系统与 DFS 之间文件的移动，这些指令的详细用法可以使用-help 指令查看，例如我们想了解 copyFromLocal 的用法：

```
2018211009@thumm01:~$ hadoop fs -help copyFromLocal
-copyFromLocal [-f] [-p] [-l] [-d] [-t <thread count>] <localsrc> ... <dst> :
  Copy files from the local file system into fs. Copying fails if the file already
  exists, unless the -f flag is given.
  Flags:
  -p              Preserves access and modification times, ownership and the mode.
  -f              Overwrites the destination if it already exists.
  -t <thread count>  Number of threads to be used, default is 1.
  -l              Allow DataNode to lazily persist the file to disk. Forces
                  replication factor of 1. This flag will result in reduced
                  durability. Use with care.
  -d              Skip creation of temporary file(<dst>._COPYING_).
```

可以看到该指令有两个必填参数，第一个参数是本地路径，第二个参数是 DFS 路径。

4.2 任务 2. 使用 MapReduce 模型进行词频统计

接下来我们要做的是使用 MapReduce 计算模型进行词频统计。

首先创建一个文件夹 input，文件夹下存放着一些文本文件：

```
2018211009@thumm01:~$ mkdir wc_input
2018211009@thumm01:~$ cd wc_input
2018211009@thumm01:~/wc_input$ echo "Hello World" > test1.txt
2018211009@thumm01:~/wc_input$ echo "Hello Hadoop" > test2.txt
2018211009@thumm01:~/wc_input$ cat test1.txt test2.txt
Hello World
Hello Hadoop
```

将 input 文件传到 DFS 中

```
2018211009@thumm01:~/input$ hadoop fs -copyFromLocal -d ~/wc_input /dsjxtjc/2018211009/
2018211009@thumm01:~/wc_input$ hadoop fs -ls /dsjxtjc/2018211009/wc_input
Found 2 items
-rw-r--r--   3 2018211009 dsjxtjc       12 2019-10-18 23:30 /dsjxtjc/2018211009/wc_input/test1.txt
-rw-r--r--   3 2018211009 dsjxtjc       13 2019-10-18 23:29 /dsjxtjc/2018211009/wc_input/test2.txt
```

在主目录下创建 word_count 文件夹，并在其中创建 WordCount.java 文件

```
2018211009@thumm01:~$ mkdir -p word_count
2018211009@thumm01:~$ cd word_count
2018211009@thumm01:~/word_count$ vim WordCount.java
```

WordCount.java 的内容如下：

```
import java.io.IOException;
import java.util.StringTokenizer;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class WordCount {
  public static class TokenizerMapper extends Mapper<Object, Text, Text, IntWritable>{

    private final static IntWritable one = new IntWritable(1);
    private Text word = new Text();
```

```
    public void map(Object key, Text value, Context context
                    ) throws IOException, InterruptedException {
      StringTokenizer itr = new StringTokenizer(value.toString());
      while (itr.hasMoreTokens()) {
        word.set(itr.nextToken());
        context.write(word, one);
      }
    }
  }

  public static class IntSumReducer extends Reducer<Text,IntWritable,Text,IntWritable> {
    private IntWritable result = new IntWritable();

    public void reduce(Text key, Iterable<IntWritable> values,
                       Context context
                       ) throws IOException, InterruptedException {
      int sum = 0;
      for (IntWritable val : values) {
        sum += val.get();
      }
      result.set(sum);
      context.write(key, result);
    }
  }

  public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();
    Job job = Job.getInstance(conf, "word count");
    job.setNumReduceTasks(3);  //@设置 reducer 的数量
    job.setJarByClass(WordCount.class);
    job.setMapperClass(TokenizerMapper.class);
    job.setCombinerClass(IntSumReducer.class);
    job.setReducerClass(IntSumReducer.class);
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);
    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));
    System.exit(job.waitForCompletion(true) ? 0 : 1);
  }
}
```

编译 Java 程序并打包为 jar 包

```
2018211009@thumm01:~/word_count$ hadoop com.sun.tools.javac.Main WordCount.java
2018211009@thumm01:~/word_count$ jar cf wc.jar WordCount*.class
```

运行 jar 包

```
2018211009@thumm01:~/word_count$ hadoop jar wc.jar WordCount /dsjxtjc/2018211009/wc_input
/dsjxtjc/2018211009/wc_output
...... //省略部分信息
2019-10-19 00:40:27,639 INFO impl.YarnClientImpl: Submitted application application_1571414388168_0010
2019-10-19 00:40:27,699 INFO mapreduce.Job: The url to track the job:
http://thumm01:8088/proxy/application_1571414388168_0010/
2019-10-19 00:40:27,700 INFO mapreduce.Job: Running job: job_1571414388168_0010
2019-10-19 00:40:34,898 INFO mapreduce.Job: Job job_1571414388168_0010 running in uber mode : false
2019-10-19 00:40:34,901 INFO mapreduce.Job:  map 0% reduce 0%
2019-10-19 00:40:39,996 INFO mapreduce.Job:  map 100% reduce 0%
2019-10-19 00:40:47,053 INFO mapreduce.Job:  map 100% reduce 100%
2019-10-19 00:40:47,067 INFO mapreduce.Job: Job job_1571414388168_0010 completed successfully
2019-10-19 00:40:47,213 INFO mapreduce.Job: Counters: 55
...... //省略部分信息
     File Output Format Counters
          Bytes Written=25
```

经过 map 和 reduce 环节后，显示 Job XXX completed successfully，表示运行成功~

接下来查看结果，将结果从 DFS 复制到本地并查看:

```
2018211009@thumm01:~/word_count$ hadoop fs -copyToLocal /dsjxtjc/2018211009/wc_output ./
2018211009@thumm01:~/word_count$ ls wc_output/
part-r-00000  _SUCCESS
2018211009@thumm01:~/word_count$ cat wc_output/part-r-00000
Hadoop     1
Hello      2
World      1
```

词频统计结束。

4.3 任务 3. 设计并实现自己的分布式文件系统

- 借鉴 DFS 和 MapReduce 的思想，用你熟悉的编程语言，自己写一个简单的 DFS+MapReduce 框架
- 利用该框架实现 **均值和方差统计** 功能，数据可以自己采集或随机生成，无固定格式要求，数据文件大小需要大于 1G，小于 5G
- 推荐通过控制台输入命令参数的方式与程序进行交互，比如
  `your_dfs_command -copyFromLocal <local_path> <dfs_path>`
- 在集群`thumm02-thumm05`机器中，请将数据块和程序中间结果放在`/home/dsjxtjc/学号`目录下
- 详细阐述你的设计，包括整体设计思想，系统框架，数据分割方案，任务分配和整合方案等细节，体现你对 DFS 和 MapReduce 思想的理解
- （加分项）有一定的任务错误处理机制（比如某台节点宕机或者出现数据块丢失）
- 具体实现可以参考附件 DFS.zip，可以在其基础上添加要求功能，也可以自己重新设计，编程语言不限。

五、 作业提交要求

- 将以上作业所用到的数据、命令、步骤截图等写入实验报告，然后连同所有代码文件一同打包成压缩文件，上交至网络学堂。
- 迟交作业一周以内，以 50%比例计分；一周以上不再计分。另一经发现抄袭情况，零分处理。
- 助教联系方式：严浩鹏（yhp18@mails.tsinghua.edu.cn）