Program 3 Report

Yangxiao Wang

Contents

1 Documentation

In this project, we are exploring the parallel technique - MapReduce. We used Inversed indexing as an example. My approach is straightforward, just map all the files with the same key (parameter) to the same reducer. And in the reducer, count the total number of the occurrence of the key. This implementation does not require combiner. When using combiner the output will have a extra 1 behind every file count (value). For example:

```
1     HDLC rfc2865.txt 22 1 rfc1122.txt 23 1
```

My opinion is that the reducer will run twice when combiner is set.

2 Source code

2.1 InvertedIndexing.java

```
1  import org.apache.hadoop.conf.*;
2  import org.apache.hadoop.fs.Path;
3  import org.apache.hadoop.io.*;
4  import org.apache.hadoop.mapred.*;
5  import org.apache.hadoop.util.*;
6
7  import java.io.IOException;
8  import java.util.*;
9
10
11 public class InvertedIndexing
12 {
13
14     public static class Map extends MapReduceBase implements Mapper<LongWritable, Text, ↩
       Text, Text>
15     {
16
17         JobConf conf;
18
19         public void configure(JobConf job)
20         {
21             this.conf = job;
22         }
23
24         public void map(LongWritable key, Text value, OutputCollector<Text, Text> output, ↩
       Reporter reporter) throws IOException
25         {
26
27             // retrieve # keywords from JobConf
28             int argc = Integer.parseInt(conf.get("argc"));
29             // put args into a String array
30             Set<String> args = new HashSet();
31             // retrieve keywords
32             for (int i = 0; i < argc; i++)
33             {
34                 args.add(conf.get("keyword" + i));
35             }
36             // get the current file name
37             FileSplit fileSplit = (FileSplit) reporter.getInputSplit();
38             String filename = "" + fileSplit.getPath().getName();
39             String lines = value.toString();
40             StringTokenizer tokenizer = new StringTokenizer(lines);
41             //collect if next token match one of the args
42             while (tokenizer.hasMoreTokens())
43             {
44                 String x = tokenizer.nextToken();
45                 if (args.contains(x))
46                 {
47                     output.collect(new Text(x), new Text(filename));
```

```
48                  }
49              }
50          }
51      }
52
53      public static class Reduce extends MapReduceBase implements Reducer<Text, Text, Text, ↩
        Text>
54      {
55
56          public void reduce(Text key, Iterator<Text> values, OutputCollector<Text, Text> ↩
        output, Reporter reporter) throws IOException
57          {
58              HashMap<String, Integer> hm = new HashMap<String, Integer>();
59              //Count the occurrence number of key in each file
60              while (values.hasNext())
61              {
62                  String name = values.next().toString();
63                  if (hm.containsKey(name))
64                  {
65                      hm.put(name, hm.get(name) + 1);
66                  }
67                  else
68                  {
69                      hm.put(name, 1);
70                  }
71              }
72              //create Comparator to sort the result by count number
73              Comparator<java.util.Map.Entry<String, Integer>> valueComparator =
74              new Comparator<java.util.Map.Entry<String, Integer>>()
75              {
76                  @Override
77                  public int compare(java.util.Map.Entry<String, Integer> e1, java.util.Map.↩
        Entry<String, Integer> e2)
78                  {
79                      return e1.getValue() - e2.getValue();
80                  }
81              };
82
83              //
84              sort the result
85              List<java.util.Map.Entry<String, Integer>> listDoc =
86              new ArrayList<java.util.Map.Entry<String, Integer>>(hm.entrySet());
87              Collections.sort(listDoc, valueComparator);
88
89              //create output string
90              StringBuilder sb = new StringBuilder();
91              for (java.util.Map.Entry<String, Integer> e : listDoc)
92              {
93                  sb.append(e.getKey());
94                  sb.append(" ");
95                  sb.append(e.getValue());
96                  sb.append(" ");
97              }
98
99              //output
100             Text docListC = new Text(sb.toString());
101             output.collect(key, docListC);
102         }
103     }
104
105     public static void main(String[] args) throws Exception
106     {
107         long time = System.currentTimeMillis();
108         JobConf conf = new JobConf(InvertedIndexing.class);
109         conf.setJobName("invertInd");
110
111         conf.setOutputKeyClass(Text.class);
112         conf.setOutputValueClass(Text.class);
113
114         conf.setMapperClass(Map.class);
115         //no need to combine because reducer already taken care of it
```

```
116         //conf.setCombinerClass(Reduce.class);
117         conf.setReducerClass(Reduce.class);
118 
119         conf.setInputFormat(TextInputFormat.class);
120         conf.setOutputFormat(TextOutputFormat.class);
121 
122         FileInputFormat.setInputPaths(conf, new Path(args[0]));
123         FileOutputFormat.setOutputPath(conf, new Path(args[1]));
124 
125         // argc maintains #keywords
126         conf.set("argc", String.valueOf(args.length - 2));
127         for (int i = 0; i < args.length - 2; i++)
128         {
129             conf.set("keyword" + i, args[i + 2]);
130         }
131 
132         JobClient.runJob(conf);
133         System.out.println("Elapsed time = " + (System.currentTimeMillis() - time) + " ms");
134     }
135 }
```

3 Execution output

3.1 Output analysis

- The performance of sequential version: 253.143 s
- The performance of parallel version: 70.517 s
- Improvement: 253.143 / 70.517 = 3.5898 times
- Cannot use diff to compare those two output because the order of the files with the same number of count is basically random. However, by comparing the result of the first two parameter HDLC and LAN and some large item with large count number, it is safe to say the outputs are the same.

3.2 Output of sequential version

```
1  HDLC    rfc2865.txt 1 rfc1122.txt 1 rfc891.txt 2 rfc907.txt 2 rfc2863.txt 3 rfc1662.↩
       txt 4
2  LAN     rfc2613.txt 1 rfc1044.txt 1 rfc4862.txt 1 rfc1123.txt 1 rfc2348.txt 1 rfc3461.↩
       txt 1 rfc1661.txt 1 rfc1155.txt 2 rfc5321.txt 2 rfc2115.txt 2 rfc1629.txt 3 rfc1559↩
       .txt 3 rfc1724.txt 3 rfc2895.txt 4 rfc1660.txt 5 rfc1213.txt 5 rfc1659.txt 5 ↩
       rfc1658.txt 5 rfc1212.txt 5 rfc1748.txt 6 rfc1694.txt 7 rfc1122.txt 10 rfc2427.txt ↩
       11 rfc950.txt 12 rfc2067.txt 17
3  PPP     rfc5531.txt 1 rfc2115.txt 1 rfc4861.txt 1 rfc1981.txt 1 rfc2427.txt 1 rfc4862.↩
       txt 1 rfc1659.txt 1 rfc4941.txt 2 rfc5036.txt 2 rfc2460.txt 3 rfc2863.txt 12 ↩
       rfc2865.txt 16 rfc1762.txt 19 rfc1994.txt 21 rfc1662.txt 22 rfc1989.txt 26 rfc1661.↩
       txt 40 rfc5072.txt 61 rfc1990.txt 72
4  TCP     rfc6152.txt 1 rfc907.txt 1 rfc919.txt 1 rfc5322.txt 1 rfc2289.txt 1 rfc4456.↩
       txt 1 rfc2067.txt 1 rfc922.txt 1 rfc868.txt 1 rfc5730.txt 1 rfc1155.txt 1 rfc1658.↩
       txt 1 rfc4941.txt 1 rfc1870.txt 1 rfc3550.txt 1 rfc2355.txt 2 rfc1044.txt 2 rfc1188↩
       .txt 2 rfc1132.txt 2 rfc1201.txt 2 rfc5065.txt 2 rfc1288.txt 2 rfc3986.txt 2 ↩
       rfc1390.txt 2 rfc894.txt 2 rfc895.txt 2 rfc1184.txt 2 rfc862.txt 3 rfc5531.txt 3 ↩
       rfc863.txt 3 rfc792.txt 3 rfc3912.txt 3 rfc3801.txt 3 rfc2895.txt 3 rfc867.txt 3 ↩
       rfc1042.txt 3 rfc866.txt 3 rfc1055.txt 3 rfc865.txt 3 rfc1356.txt 3 rfc1034.txt 5 ↩
       rfc1772.txt 5 rfc864.txt 5 rfc959.txt 5 rfc3551.txt 6 rfc4862.txt 7 rfc1939.txt 8 ↩
       rfc2741.txt 8 rfc2920.txt 9 rfc4861.txt 9 rfc854.txt 10 rfc2865.txt 10 rfc5321.txt ↩
       10 rfc2132.txt 11 rfc791.txt 12 rfc2460.txt 12 rfc1035.txt 12 rfc1981.txt 18 ↩
       rfc1006.txt 24 rfc1191.txt 25 rfc1213.txt 33 rfc1002.txt 38 rfc1123.txt 41 rfc5734.↩
       txt 42 rfc5036.txt 58 rfc5681.txt 83 rfc1001.txt 123 rfc4271.txt 126 rfc1122.txt ↩
       221 rfc793.txt 278
5  UDP     rfc868.txt 1 rfc1629.txt 1 rfc2348.txt 1 rfc2132.txt 1 rfc1055.txt 1 rfc950.↩
       txt 1 rfc5531.txt 2 rfc791.txt 2 rfc4862.txt 2 rfc1034.txt 2 rfc3411.txt 2 rfc2453.↩
       txt 2 rfc867.txt 3 rfc862.txt 3 rfc1981.txt 3 rfc1350.txt 3 rfc863.txt 3 rfc792.txt↩
        3 rfc1191.txt 3 rfc866.txt 3 rfc865.txt 3 rfc2895.txt 3 rfc864.txt 4 rfc3551.txt 4↩
        rfc4502.txt 5 rfc2131.txt 5 rfc768.txt 6 rfc2460.txt 8 rfc5036.txt 10 rfc951.txt ↩
       11 rfc3417.txt 12 rfc1035.txt 13 rfc3550.txt 15 rfc1213.txt 19 rfc1542.txt 21 ↩
       rfc2865.txt 24 rfc1123.txt 25 rfc1001.txt 33 rfc1002.txt 50 rfc1122.txt 65
```

3.3 Output of parallel version

```
1  HDLC    rfc2865.txt 1 rfc1122.txt 1 rfc891.txt 2 rfc907.txt 2 rfc2863.txt 3 rfc1662.↩
       txt 4
2  LAN rfc2613.txt 1 rfc1044.txt 1 rfc4862.txt 1 rfc1123.txt 1 rfc2348.txt 1 rfc3461.txt ↩
       1 rfc1661.txt 1 rfc1155.txt 2 rfc5321.txt 2 rfc2115.txt 2 rfc1629.txt 3 rfc1559.txt↩
        3 rfc1724.txt 3 rfc2895.txt 4 rfc1660.txt 5 rfc1213.txt 5 rfc1659.txt 5 rfc1658.↩
       txt 5 rfc1212.txt 5 rfc1748.txt 6 rfc1694.txt 7 rfc1122.txt 10 rfc2427.txt 11 ↩
       rfc950.txt 12 rfc2067.txt 17
3  PPP rfc5531.txt 1 rfc2115.txt 1 rfc4861.txt 1 rfc1981.txt 1 rfc2427.txt 1 rfc4862.txt ↩
       1 rfc1659.txt 1 rfc4941.txt 2 rfc5036.txt 2 rfc2460.txt 3 rfc2863.txt 12 rfc2865.↩
       txt 16 rfc1762.txt 19 rfc1994.txt 21 rfc1662.txt 22 rfc1989.txt 26 rfc1661.txt 40 ↩
       rfc5072.txt 61 rfc1990.txt 72
4  TCP rfc6152.txt 1 rfc907.txt 1 rfc919.txt 1 rfc5322.txt 1 rfc2289.txt 1 rfc4456.txt 1 ↩
       rfc2067.txt 1 rfc922.txt 1 rfc868.txt 1 rfc5730.txt 1 rfc1155.txt 1 rfc1658.txt 1 ↩
       rfc4941.txt 1 rfc1870.txt 1 rfc3550.txt 1 rfc2355.txt 2 rfc1044.txt 2 rfc1188.txt 2↩
```

```
         rfc1132.txt 2 rfc1201.txt 2 rfc5065.txt 2 rfc1288.txt 2 rfc3986.txt 2 rfc1390.txt ↩
        2 rfc894.txt 2 rfc895.txt 2 rfc1184.txt 2 rfc862.txt 3 rfc5531.txt 3 rfc792.txt 3 ↩
        rfc863.txt 3 rfc3912.txt 3 rfc3801.txt 3 rfc2895.txt 3 rfc867.txt 3 rfc1042.txt 3 ↩
        rfc866.txt 3 rfc1055.txt 3 rfc865.txt 3 rfc1356.txt 3 rfc1034.txt 5 rfc1772.txt 5 ↩
        rfc959.txt 5 rfc864.txt 5 rfc3551.txt 6 rfc4862.txt 7 rfc1939.txt 8 rfc2741.txt 8 ↩
        rfc2920.txt 9 rfc4861.txt 9 rfc854.txt 10 rfc2865.txt 10 rfc5321.txt 10 rfc2132.txt↩
         11 rfc791.txt 12 rfc2460.txt 12 rfc1035.txt 12 rfc1981.txt 18 rfc1006.txt 24 ↩
        rfc1191.txt 25 rfc1213.txt 33 rfc1002.txt 38 rfc1123.txt 41 rfc5734.txt 42 rfc5036.↩
        txt 58 rfc5681.txt 83 rfc1001.txt 123 rfc4271.txt 126 rfc1122.txt 221 rfc793.txt ↩
        278
  UDP rfc868.txt 1 rfc1629.txt 1 rfc2348.txt 1 rfc2132.txt 1 rfc1055.txt 1 rfc950.txt 1 ↩
        rfc5531.txt 2 rfc791.txt 2 rfc4862.txt 2 rfc1034.txt 2 rfc3411.txt 2 rfc2453.txt 2 ↩
        rfc862.txt 3 rfc867.txt 3 rfc1981.txt 3 rfc1350.txt 3 rfc863.txt 3 rfc792.txt 3 ↩
        rfc1191.txt 3 rfc866.txt 3 rfc865.txt 3 rfc2895.txt 3 rfc864.txt 4 rfc3551.txt 4 ↩
        rfc4502.txt 5 rfc2131.txt 5 rfc768.txt 6 rfc2460.txt 8 rfc5036.txt 10 rfc951.txt 11↩
         rfc3417.txt 12 rfc1035.txt 13 rfc3550.txt 15 rfc1213.txt 19 rfc1542.txt 21 rfc2865↩
        .txt 24 rfc1123.txt 25 rfc1001.txt 33 rfc1002.txt 50 rfc1122.txt 65
```

4 Discussions

1. File distribution over a cluster system

   MPI is Message Passing Interface, it does not need a file system to store its data. And the data is sent to another node to be computed.

   On the other hand, MapReduce is usually used with Hadoop Distributed File System. And the data is stored in local storage on each data node.

2. Collective/Reductive operation to create inverted indexing

   The implementation with MPI will be:

   - read all data, distribute the data to each node.
   - each node will compute and find the number of keywords' occurrence in their portion of data.
   - share the whole results and do the reduction.

   The biggest problem with MPI when doing inverted indexing is that it uses network to transfer data. When the size of files is large, the MPI's performance will not be ideal.

3. Amount of boilerplate code

   Comparing to MapReduce, MPI would have more boilerplate code like send, receive, and parse data. And MapReduce only need implement the Map class and Reducer class.

4. Anticipated execution performance

   Again, the limitation of MPI is the network, when the size of data/file is large and the performance could be slower than MapReduce. However, if the data is not too large, using MPI could be more efficient because the Hadoop's fault tolerance system can slow down the process.

5. Fault tolerance; recovery from a crash

   MPI support checkpoint to restart from the checkpoint if anything goes wrong. It does not have Message logging techniques, data Reliability and network fault tolerance, User directed and communicator driven fault tolerance. Basically, developer can set multiple checkpoints before passing data or during iteration. However, the data could be still lost if the network is not stable.

Hadoop has its own built-in fault tolerance and fault compensation capabilities. Every data block has a copy that is stored on other servers. And it also generate logs during the execution process.

5 Lab Sessions 3

5.1 Execution output

```
[wyxiao_css534@cssmpi1 wordcount_2.0]$ hadoop jar wordcount.jar WordCount input output
18/11/08 18:37:12 WARN mapred.JobClient: Use GenericOptionsParser for parsing the arguments. Applications should impleme
nt Tool for the same.
18/11/08 18:37:12 INFO mapred.FileInputFormat: Total input paths to process : 2
18/11/08 18:37:13 INFO mapred.JobClient: Running job: job_201811071847_0013
18/11/08 18:37:14 INFO mapred.JobClient:  map 0% reduce 0%
18/11/08 18:37:21 INFO mapred.JobClient:  map 33% reduce 0%
18/11/08 18:37:23 INFO mapred.JobClient:  map 66% reduce 0%
18/11/08 18:37:28 INFO mapred.JobClient:  map 100% reduce 0%
18/11/08 18:37:30 INFO mapred.JobClient:  map 100% reduce 11%
18/11/08 18:37:33 INFO mapred.JobClient:  map 100% reduce 22%
18/11/08 18:37:39 INFO mapred.JobClient:  map 100% reduce 100%
18/11/08 18:37:41 INFO mapred.JobClient: Job complete: job_201811071847_0013
18/11/08 18:37:41 INFO mapred.JobClient: Counters: 18
18/11/08 18:37:41 INFO mapred.JobClient:   Map-Reduce Framework
18/11/08 18:37:41 INFO mapred.JobClient:     Combine output records=6
18/11/08 18:37:41 INFO mapred.JobClient:     Spilled Records=12
18/11/08 18:37:41 INFO mapred.JobClient:     Reduce input records=6
18/11/08 18:37:41 INFO mapred.JobClient:     Reduce output records=5
18/11/08 18:37:41 INFO mapred.JobClient:     Map input records=2
18/11/08 18:37:41 INFO mapred.JobClient:     Map output records=8
18/11/08 18:37:41 INFO mapred.JobClient:     Map output bytes=82
18/11/08 18:37:41 INFO mapred.JobClient:     Reduce shuffle bytes=46
18/11/08 18:37:41 INFO mapred.JobClient:     Combine input records=8
18/11/08 18:37:41 INFO mapred.JobClient:     Map input bytes=50
18/11/08 18:37:41 INFO mapred.JobClient:     Reduce input groups=5
18/11/08 18:37:41 INFO mapred.JobClient:   FileSystemCounters
18/11/08 18:37:41 INFO mapred.JobClient:     HDFS_BYTES_READ=54
18/11/08 18:37:41 INFO mapred.JobClient:     FILE_BYTES_WRITTEN=266
18/11/08 18:37:41 INFO mapred.JobClient:     FILE_BYTES_READ=79
18/11/08 18:37:41 INFO mapred.JobClient:     HDFS_BYTES_WRITTEN=41
18/11/08 18:37:41 INFO mapred.JobClient:   Job Counters
18/11/08 18:37:41 INFO mapred.JobClient:     Launched map tasks=3
18/11/08 18:37:41 INFO mapred.JobClient:     Launched reduce tasks=1
18/11/08 18:37:41 INFO mapred.JobClient:     Rack-local map tasks=3
[wyxiao_css534@cssmpi1 wordcount_2.0]$
```

Figure 1: MapReduce execution

```
[wyxiao_css534@cssmpi1 ~]$ hadoop fs -ls /user/wyxiao_css534/output
Found 2 items
drwxr-xr-x   - wyxiao_css534 wyxiao_css534          0 2018-11-07 21:37 /user/wyxiao_css534/output/_logs
-rw-r--r--   3 wyxiao_css534 wyxiao_css534       2283 2018-11-07 21:38 /user/wyxiao_css534/output/part-00000
```

Figure 2: /user/yourAccount/output

```
[wyxiao_css534@cssmpi1 output]$ cat part-00000
Bye     1
Goodbye 1
Hadoop  2
He      1
Hello   2
World   2
he      2
```

Figure 3: part-00000