LAB 3: MAPREDUCE

Q1: How many times does the word Sherlock appear in the file?

The propose of part 1 is to statistic how many times a word occurs in the text. When we are using Hadoop to do the distributed calculation, what we should consider are the MAP job, REDUCE job and a MAIN function which invokes the program and configure the nodes. MAP is responsible to spilt all the word in the text and emit each word with a key-value pair, like ("word", 1) and send word-iterationList like ("word", 1, 1, 1…) to REDUCE nodes. REDUCE is responsible for aggregating the iteration list for each word and generating a ("word", count) pair for each unique word. Notice that a word is only sent to a REDUCE node. So the result of the REDUCE is the word-wordCount pair. To find the times that "Sherlock" appears, we only need to check the file and find the count, which is 345.

In *TokenizerMapper.java,*

StringTokenizer itr = new StringTokenizer(value.toString(), "-- \t\n\r\f,.:;?![]()'\"");

means spilt the text into a list if it meets the following syntax: "-- \t\n\r\f,.:;?![]()'\"".

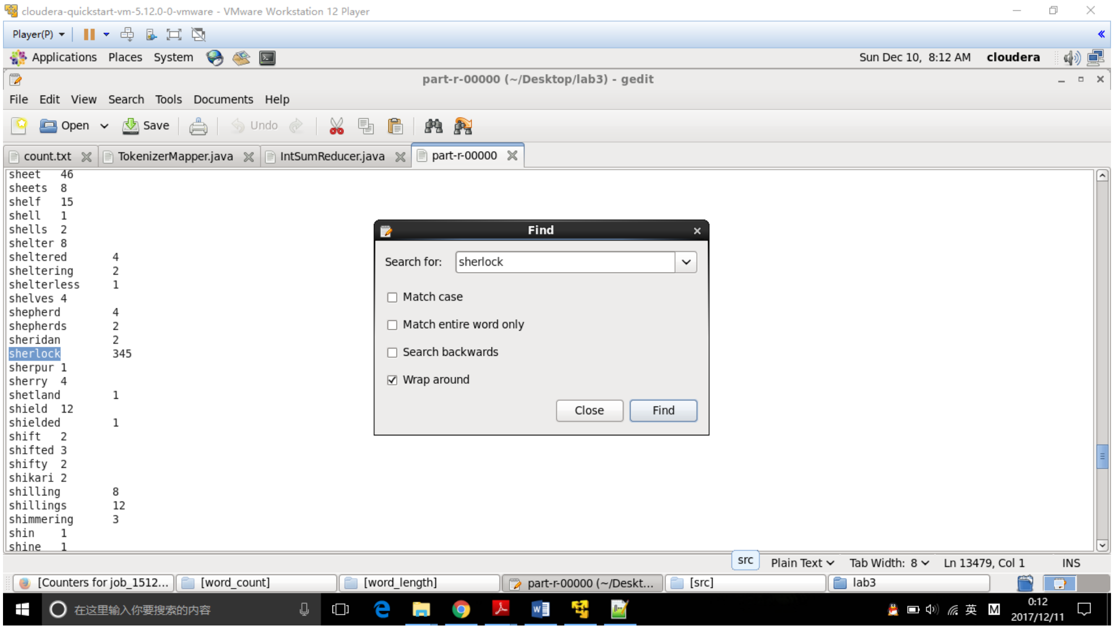
In *IntSumReducer.java,*

sum += value.get();

value.get() is to turn IntWritable type into int type; add the element of iterable list to sum.

context.write(key, result);

emit the (word, count) pair.



```
sheet   46
sheets  8
shelf   15
shell   1
shells  2
shelter 8
sheltered       4
sheltering      2
shelterless     1
shelves 4
shepherd        4
shepherds       2
sheridan        2
sherlock        345
sherpur 1
sherry  4
shetland        1
shield  12
shielded        1
shift   2
shifted 3
shifty  2
shikari 2
shilling        8
shillings       12
shimmering      3
shin    1
shine   1
```

Find

Search for: sherlock

☐ Match case

☐ Match entire word only

☐ Search backwards

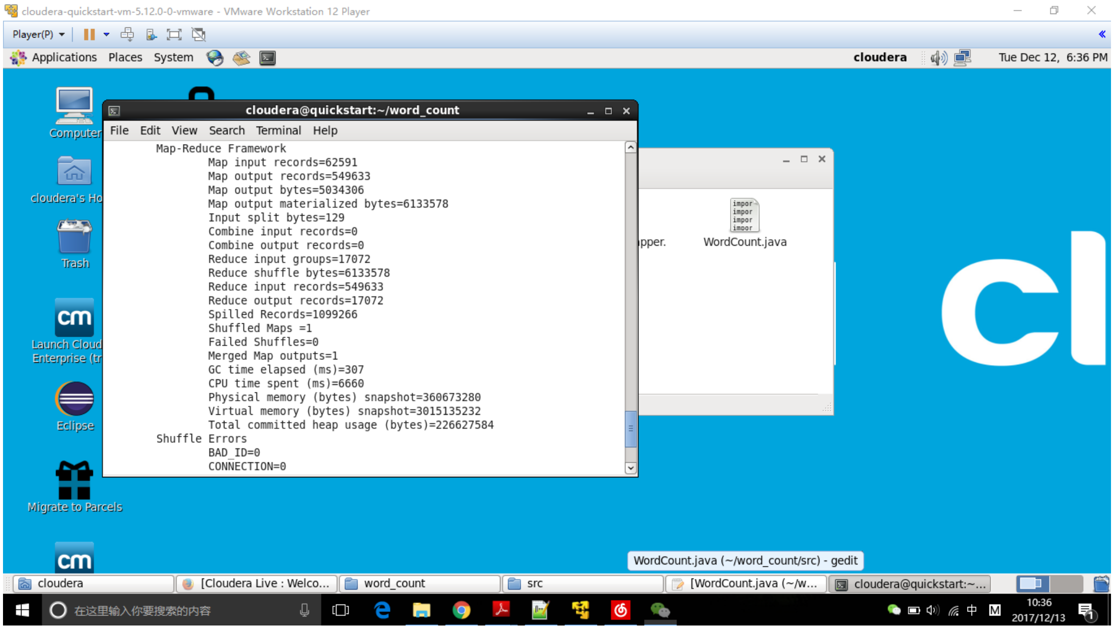
☑ Wrap around

Close Find

Q2: The Hadoop job outputs a set of statistics after the completion. What was the number of key-value pairs generated by the Mapper, the number of unique inputs to the Reducer, and the final number of records emitted by the Reducer? Based on these values, answer the following two questions: How many words has the document in total? How many unique words does the document have?

According to our MAP method, which emits ("words", 1) for each word, the number of key-value pairs generated by the Mapper is actually the total number of words in the text. From the screenshot, we can see the value is 549633. The number of unique inputs to the Reducer is the Reduce input group which is 17072. The final number of records emitted by the Reducer is 17072 since the reducer adds the occurrence times of each word, so it is also equal to the number of unique words.

So base on the analysis above, the document has 549633 words in total. There are 17072 unique words in the document.

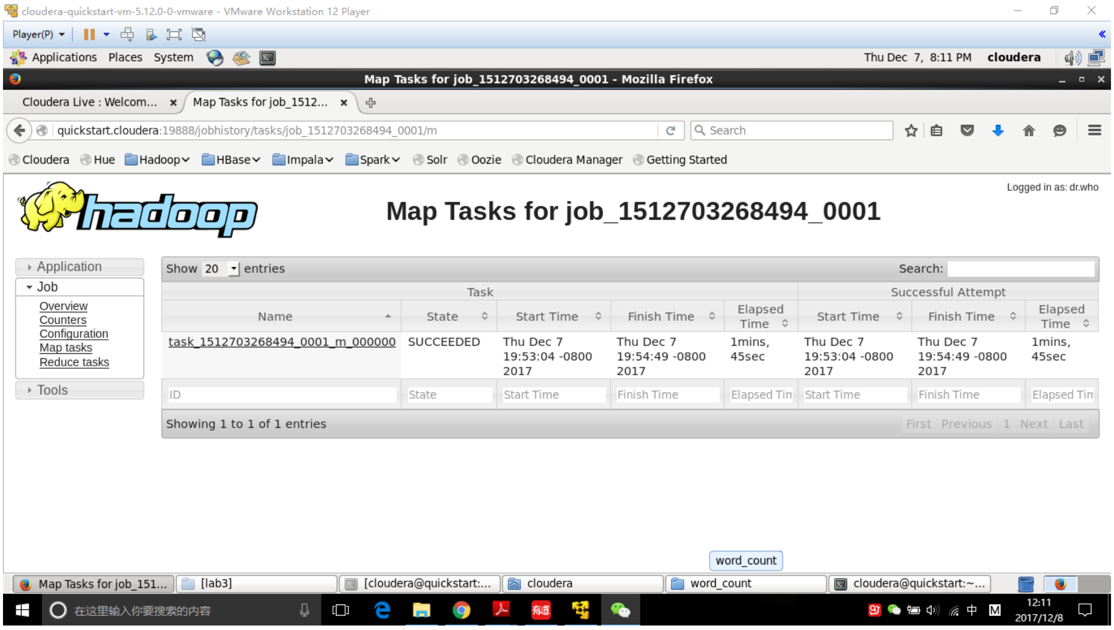


cloudera-quickstart-vm-5.12.0-0-vmware - VMware Workstation 12 Player
Player(P)
Applications Places System
cloudera
Tue Dec 12, 6:36 PM
cloudera@quickstart:~/word_count
File Edit View Search Terminal Help
WordCount.java
WordCount.java (~/word_count/src) - gedit
cloudera
[Cloudera Live : Welco...
word_count
src
[WordCount.java (~/w...
cloudera@quickstart:~...
在这里输入你要搜索的内容
10:36
2017/12/13

```
        Map-Reduce Framework
                Map input records=62591
                Map output records=549633
                Map output bytes=5034306
                Map output materialized bytes=6133578
                Input split bytes=129
                Combine input records=0
                Combine output records=0
                Reduce input groups=17072
                Reduce shuffle bytes=6133578
                Reduce input records=549633
                Reduce output records=17072
                Spilled Records=1099266
                Shuffled Maps =1
                Failed Shuffles=0
                Merged Map outputs=1
                GC time elapsed (ms)=307
                CPU time spent (ms)=6660
                Physical memory (bytes) snapshot=360673280
                Virtual memory (bytes) snapshot=3015135232
                Total committed heap usage (bytes)=226627584
        Shuffle Errors
                BAD_ID=0
                CONNECTION=0
```

Q3: How many Map and Reduce tasks does your MapReduce job have? Can you explain the difference with the Sherlock job in Part 1?

Since we added *"job.setNumReduceTasks(2);"* in our "main class", which means we set two Reduce nodes to do the job. So we have 1 Map task and 2 Reduce tasks. Notice that the task in 2 Reduce nodes is executed parallel and each node is allocated about half of the total output of Map job. So compared with Sherlock job which only has 1 Reduce node, the Reduce job is faster in time in the case of the same Reduce input.

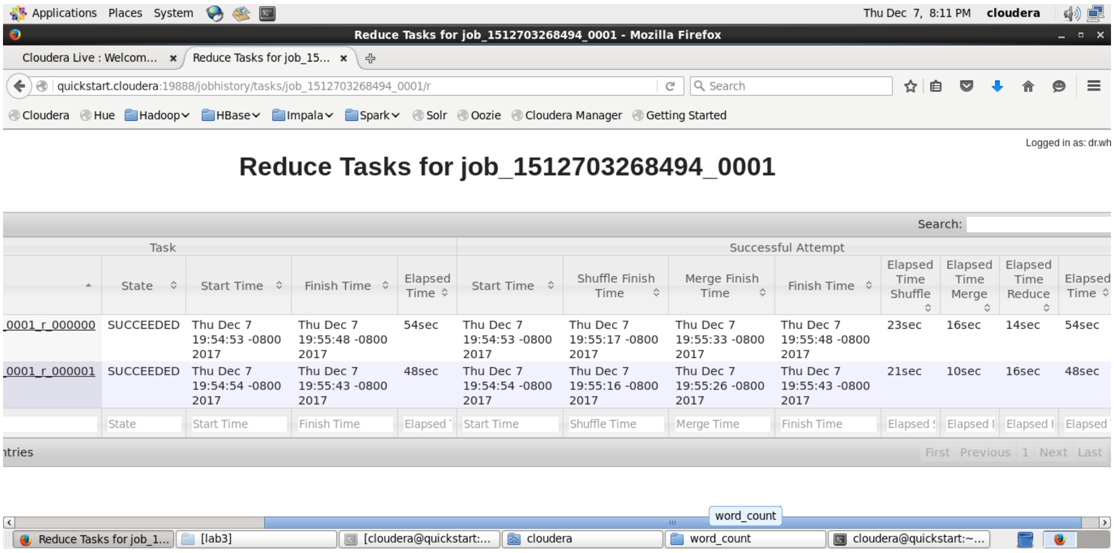

cloudera-quickstart-vm-5.12.0-0-vmware - VMware Workstation 12 Player
Player(P)
Applications Places System
Thu Dec 7, 8:11 PM cloudera
Map Tasks for job_1512703268494_0001 - Mozilla Firefox
Cloudera Live : Welcom...
Map Tasks for job_1512...
quickstart.cloudera:19888/jobhistory/tasks/job_1512703268494_0001/m
Search
Cloudera Hue Hadoop HBase Impala Spark Solr Oozie Cloudera Manager Getting Started
Logged in as: dr.who
Application
Job
Overview
Counters
Configuration
Map tasks
Reduce tasks
Tools
word_count
Map Tasks for job_151...
[lab3]
[cloudera@quickstart:...
cloudera
word_count
cloudera@quickstart:~...
在这里输入你要搜索的内容
12:11
2017/12/8

Map Tasks for job_1512703268494_0001

Show 20 entries Search:

Task					Successful Attempt		
Name	State	Start Time	Finish Time	Elapsed Time	Start Time	Finish Time	Elapsed Time
task_1512703268494_0001_m_000000	SUCCEEDED	Thu Dec 7 19:53:04 -0800 2017	Thu Dec 7 19:54:49 -0800 2017	1mins, 45sec	Thu Dec 7 19:53:04 -0800 2017	Thu Dec 7 19:54:49 -0800 2017	1mins, 45sec
ID	State	Start Time	Finish Time	Elapsed Tin	Start Time	Finish Time	Elapsed Tin

Showing 1 to 1 of 1 entries First Previous 1 Next Last


Applications Places System
Thu Dec 7, 8:11 PM cloudera
Reduce Tasks for job_1512703268494_0001 - Mozilla Firefox
Cloudera Live : Welcom...
Reduce Tasks for job_15...
quickstart.cloudera:19888/jobhistory/tasks/job_1512703268494_0001/r
Search
Cloudera Hue Hadoop HBase Impala Spark Solr Oozie Cloudera Manager Getting Started
Logged in as: dr.wh
word_count
Reduce Tasks for job_1...
[lab3]
[cloudera@quickstart:...
cloudera
word_count
cloudera@quickstart:~...

Reduce Tasks for job_1512703268494_0001

Search:

Task					Successful Attempt							
	State	Start Time	Finish Time	Elapsed Time	Start Time	Shuffle Finish Time	Merge Finish Time	Finish Time	Elapsed Time Shuffle	Elapsed Time Merge	Elapsed Time Reduce	Elapsed Time
_0001_r_000000	SUCCEEDED	Thu Dec 7 19:54:53 -0800 2017	Thu Dec 7 19:55:48 -0800 2017	54sec	Thu Dec 7 19:54:53 -0800 2017	Thu Dec 7 19:55:17 -0800 2017	Thu Dec 7 19:55:33 -0800 2017	Thu Dec 7 19:55:48 -0800 2017	23sec	16sec	14sec	54sec
_0001_r_000001	SUCCEEDED	Thu Dec 7 19:54:54 -0800 2017	Thu Dec 7 19:55:43 -0800 2017	48sec	Thu Dec 7 19:54:54 -0800 2017	Thu Dec 7 19:55:16 -0800 2017	Thu Dec 7 19:55:26 -0800 2017	Thu Dec 7 19:55:43 -0800 2017	21sec	10sec	16sec	48sec
	State	Start Time	Finish Time	Elapsed	Start Time	Shuffle Time	Merge Time	Finish Time	Elapsed	Elapsed	Elapsed	Elapsed

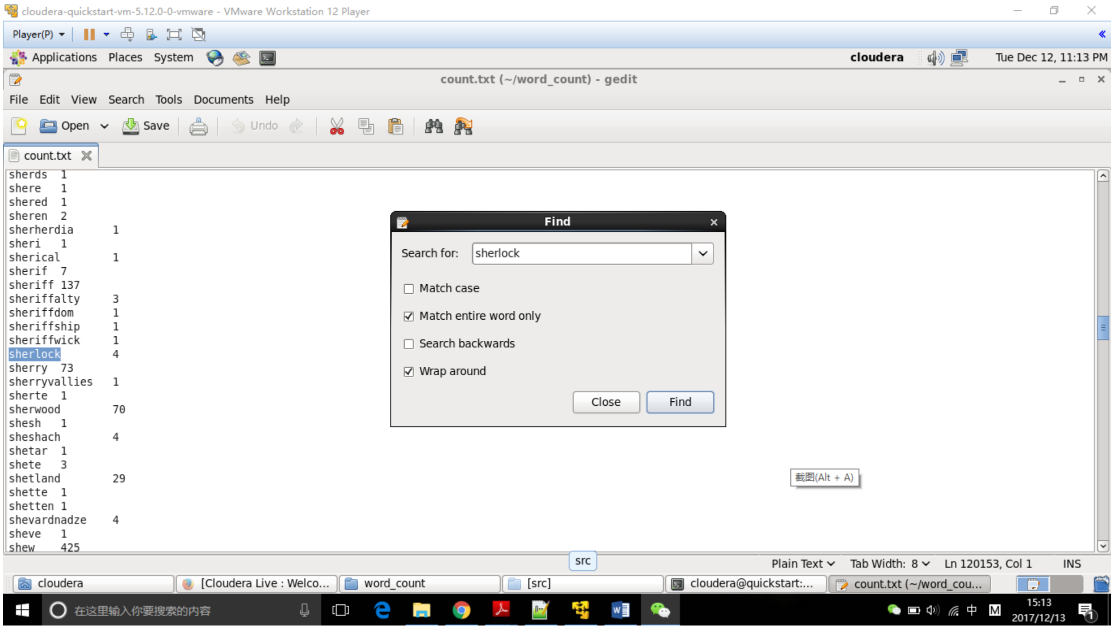
ıtries First Previous 1 Next Last

Q4: Can you see a clear pattern on how Hadoop partitions the keys among multiple reducers? Does it make it easier or harder the problem of manually retrieving information about a specific key? E.g. Answering questions such as "How many times the word Sherlock appears in the text?"

I can’t see a clear pattern on how Hadoop partitions the keys. All pairs of a key are allocated to one reducer. It is the internal mechanism of Hadoop to handle which key is distributed to which Reduce node and it should be balance for each node.

Since there are two Reduce nodes to handle the aggregation process and the Reduce tasks is parallel, I think it make it easier to handle the problem.

Sherlock appears 4 times in the text.



count.txt (~/word_count) - gedit

```
sherds	1
shere	1
shered	1
sheren	2
sherherdia	1
sheri	1
sherical	1
sherif	7
sheriff	137
sheriffalty	3
sheriffdom	1
sheriffship	1
sheriffwick	1
sherlock	4
sherry	73
sherryvallies	1
sherte	1
sherwood	70
shesh	1
sheshach	4
shetar	1
shete	3
shetland	29
shette	1
shetten	1
shevardnadze	4
sheve	1
shew	425
```

Find

Search for: sherlock

☐ Match case

☑ Match entire word only

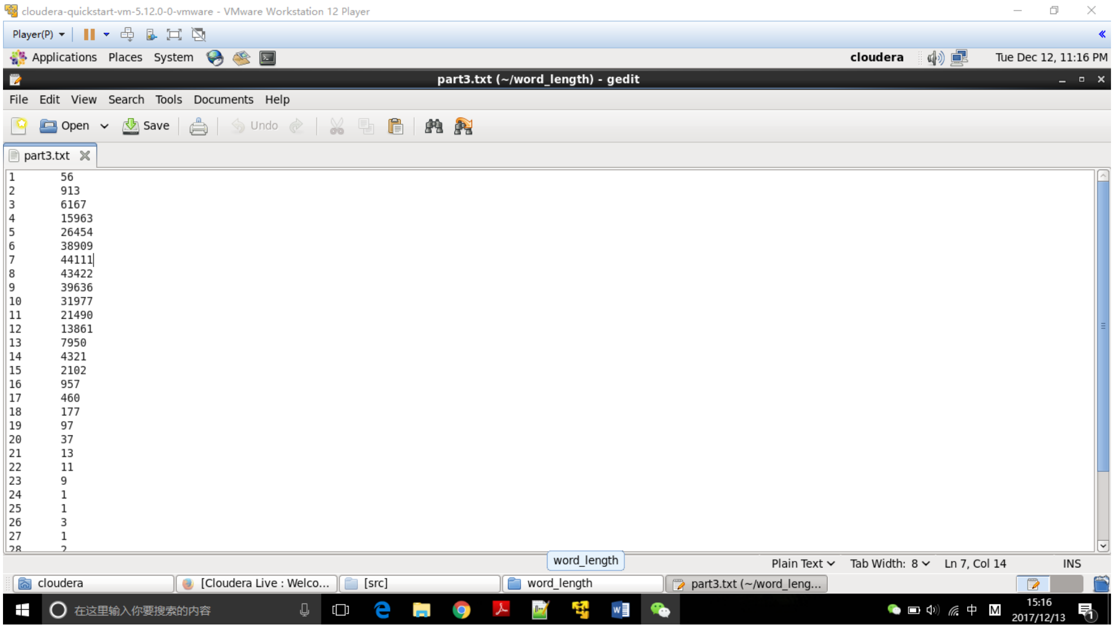
☐ Search backwards

☑ Wrap around

Plain Text Tab Width: 8 Ln 120153, Col 1 INS

Q5: What is the most common word length for the onehundredM.txt dataset? Briefly explain the Mapper and Reducer code that you have implemented.

According to the screenshot below, we find that the most common word length for the onehundredM.txt dataset is 7.



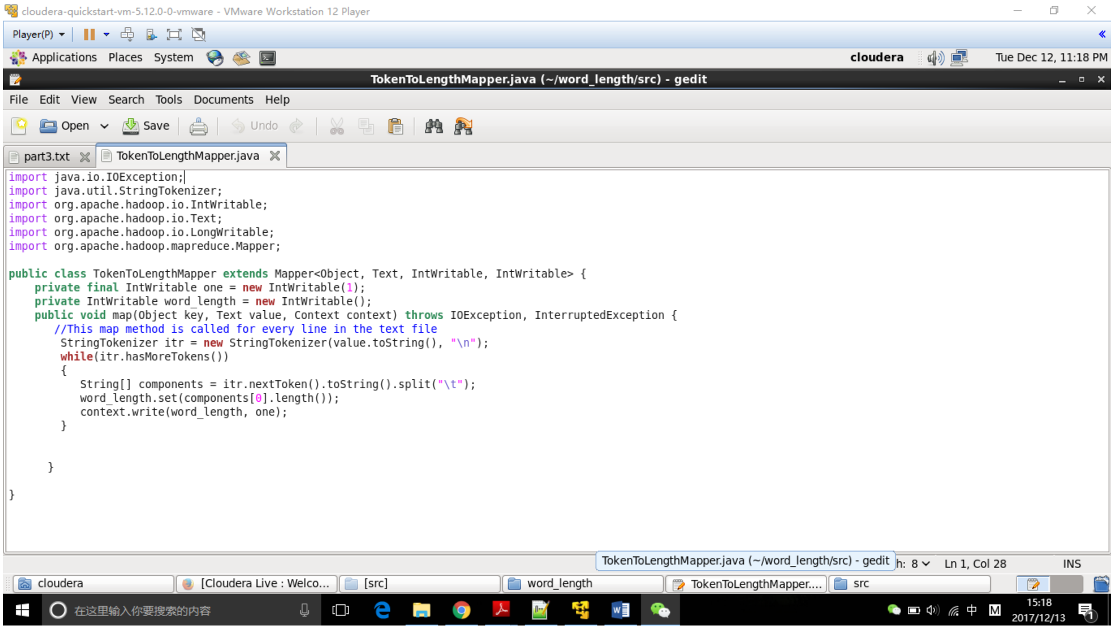
part3.txt (~/word_length) - gedit

```
1	56
2	913
3	6167
4	15963
5	26454
6	38909
7	44111
8	43422
9	39636
10	31977
11	21490
12	13861
13	7950
14	4321
15	2102
16	957
17	460
18	177
19	97
20	37
21	13
22	11
23	9
24	1
25	1
26	3
27	1
28	2
```

Plain Text Tab Width: 8 Ln 7, Col 14 INS

cloudera-quickstart-vm-5.12.0-0-vmware - VMware Workstation 12 Player
Player(P)
Applications Places System
cloudera
Tue Dec 12, 11:16 PM
File Edit View Search Tools Documents Help
Open
Save
Undo
part3.txt
word_length
cloudera
[Cloudera Live : Welco...
[src]
word_length
part3.txt (~/word_leng...
在这里输入你要搜索的内容
15:16
2017/12/13

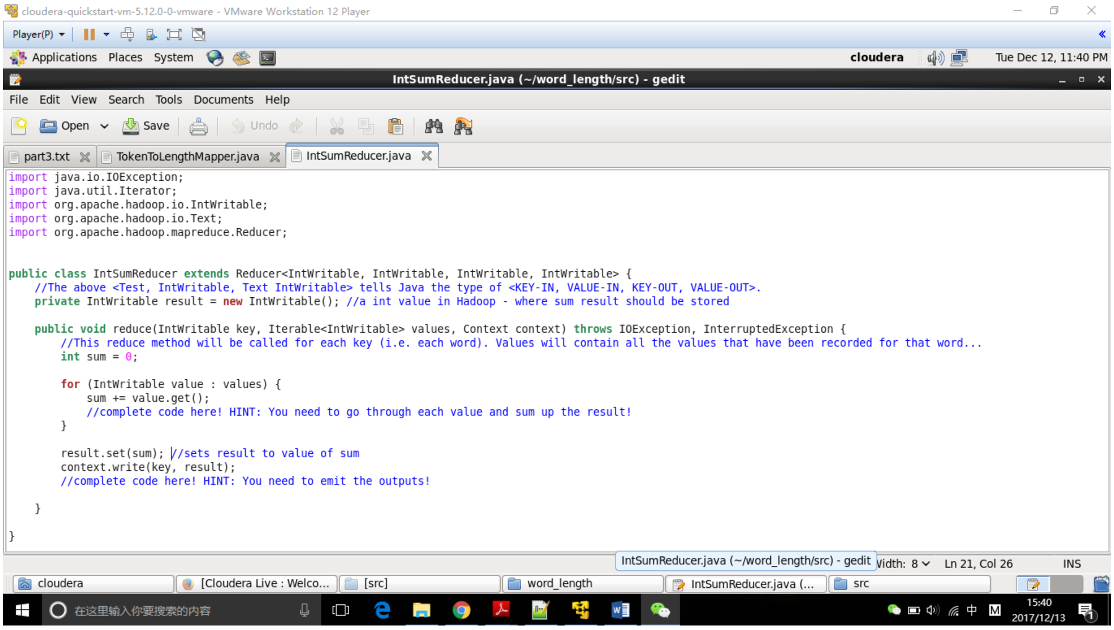
The Mapper code is shown below:



```
import java.io.IOException;
import java.util.StringTokenizer;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.mapreduce.Mapper;

public class TokenToLengthMapper extends Mapper<Object, Text, IntWritable, IntWritable> {
    private final IntWritable one = new IntWritable(1);
    private IntWritable word_length = new IntWritable();
    public void map(Object key, Text value, Context context) throws IOException, InterruptedException {
       //This map method is called for every line in the text file
        StringTokenizer itr = new StringTokenizer(value.toString(), "\n");
        while(itr.hasMoreTokens())
        {
           String[] components = itr.nextToken().toString().split("\t");
           word_length.set(components[0].length());
           context.write(word_length, one);
        }


      }

}
```

The input of Map is the document: count.txt, which generated in part 2. So the input data will be <*Object, Text*>. Then we want to emit pair for each unique word, which is <*word_length, 1*>. So the output key and value are both *IntWritable.* The format of the file has each pair in a line and the key (unique word) and value (how many times the word occurs) is divided by "\t". After we scan the whole file, the value is the whole text. Then we spilt each line through *StringTokenizer* and for each line, we spilt it to a string array. In this case, components[0] represents the word, components[1] represents the times of the words occur. Then we acquire the length of the word and emit the pair (*word_length, 1*).

The Reduce code is shown below:



```
import java.io.IOException;
import java.util.Iterator;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;


public class IntSumReducer extends Reducer<IntWritable, IntWritable, IntWritable, IntWritable> {
    //The above <Test, IntWritable, Text IntWritable> tells Java the type of <KEY-IN, VALUE-IN, KEY-OUT, VALUE-OUT>.
    private IntWritable result = new IntWritable(); //a int value in Hadoop - where sum result should be stored

    public void reduce(IntWritable key, Iterable<IntWritable> values, Context context) throws IOException, InterruptedException {
        //This reduce method will be called for each key (i.e. each word). Values will contain all the values that have been recorded for that word...
        int sum = 0;

        for (IntWritable value : values) {
            sum += value.get();
            //complete code here! HINT: You need to go through each value and sum up the result!
        }

        result.set(sum); //sets result to value of sum
        context.write(key, result);
        //complete code here! HINT: You need to emit the outputs!

    }

}
```

Since the type that Map emit is <IntWritable, IntWritable>, the input key should be IntWritable and the value is a IntWritable list. The output of Reduce is also pairs < IntWritable, IntWritable >. Since

Reduce method is called for each key, what we need to do is to add the 1 of the list and emit the pair (word_length, sum_of _uniqueWord_has_the_same_length).