Assignmet 02

==========

Question 1)

----------------

Commands executed are in order :-

i. hadoop fs -mkdir /user/JigBD0008/wordcount

ii.hadoop fs -mkdir /user/JigBD0008/wordcount/input

iii.  hadoop fs -copyFromLocal wordcountproblem.txt /user/JigBD0008/wordcount/input

iv. hadoop jar wc.jar WordCount /user/JigBD0008/wordcount/input/wordcountproblem.txt /user/JigBD0008/wordcount/output

v.  hadoop fs -cat /user/JigBD0008/wordcount/output/part-00000

Output of the above command is :-

& 26  
(Anatomy 1  
(Configuration 2  
(Configuring 13  
(Data 13  
(Input 6  
(Managed 13  
(Scheduling, 6  
(Schema 13  
(Sorting 13  
(TextOutput, 6  
- 24  
A 34  
According 17  
Account 2  
Addresses 2  
Aggregating, 13  
Altering 13  
Anatomy 1  
And 30  
Apache 68  
Appending 13  
Architecture, 6  
Archives 1  
BIG 51  
Benchmarking 2  
BiG 1  
Big 57  
Binary 6  
BinaryOutPut, 6  
Buckets, 13  
Cafarella. 17  
Cluster 10  
ClusterCluster 1  
Coherency 1  
Combiners, 6  
Command 1  
Comparison 13  
Concept 24  
Concepts, 1  
Concurrency 6  
Configuration 2  
Configuration, 2  
Copying 1  
Creation), 2  
Custom 13  
Cutting 17  
DATA 17  
DISTCP, 1  
Daemon 4  
Data 118  
Data! 51  
Data, 13  
Data/Code 6  
Database 19  
Databaseoutput) 1  
Databaseoutput)Hadoop 5  
Defined 13  
Design 1  
Distributed 6  
Doug 17  
Doug�s 17  
Dropping 13  
Environment 2  
Error 6  
Execution 6  
External 13  
Fault 6  
File 9  
Flow 1  
Formats 12  
Formats, 13  
Framework 6  
Functional 36  
Functions), 13  
Functions, 13  
Google 17  
HDFS 1  
HDFS, 1  
Hadoop 394  
Hadoop! 17  
Hadoop. 17  
HadoopBig 45  
Handling), 6  
Hive 40  
Hive, 26  
HiveInstalling 12  
HiveQL 13  
However 17  
IT 17  
If 17  
Imperative 6  
Important 2  
Importing 13  
In 17  
Indexes) 1  
Indexes), 12  
Input 12  
Input, 12  
Inputs, 6  
Installation, 2  
Installing 1  
Interface, 2  
It 51  
J. 17  
Java 18  
Join 13  
Joings 13  
Line 1  
Management, 2  
Map 26  
Map/Reduce 13  
MapReduce 23  
Mappers, 6  
MetaStore), 13  
Michael 17  
Model), 1  
Multiple 12  
Operators 13  
Other 2  
OutPut 6  
Output), 6  
Outputs, 6  
Parallel 1  
Partitioners, 6  
Partitions 13  
Ports, 2  
Problem? 17  
Programming 6  
Programming, 6  
Properties, 4  
Query), 13  
Querying 13  
Read 13  
Read, 1  
Records, 6  
Reduce 26  
Reducers, 6  
Roots, 6  
Running 13  
SSH 2  
Schema 13  
Scripts, 13  
Security, 2  
Services, 13  
Settings, 2  
Setup 2  
Solution 17  
Specification, 2  
Splits 6  
Storage 13  
Subqueries 13  
Synchronization, 6  
System, 6  
Systems, 1  
Table, 13  
Tables 26  
Tables), 13  
Tables, 26  
Text 6  
The 40  
Though 17  
Traditional 13  
Transactions 13  
Types, 25  
Updates, 13  
User 15  
Versus 13  
Views, 13  
Wikipedia, 17  
Worldwide 17  
Write, 14  
a 123  
across? 17  
after 17  
ago, 17  
all 17  
almost 34  
amount 17  
an 34  
and 348  
anxiety 17  
any 17  
applications 17  
applications. 17  
applications.� 17  
applies 17  
are 17  
as 17  
based 17  
basically 17  
become 17  
becomes 17  
becoming 68  
been 17  
being 17  
biggest 17  
by 51  
can 17  
charming 17  
co-location, 6  
collection 17  
community 17  
companies 51  
company 34  
complex 17  
complexity 17  
concepts 17  
constructed 17  
contributors. 17  
data 91  
data, 17  
data-intensive 17  
data. 34  
data.What 17  
database 17  
developed 17  
developed, 17  
difficult 17  
distributed 17  
don�t 17  
elephant 17  
elephant!, 1  
elephant!BiG 16  
enormous 17  
evaluating 17  
even 17  
ever-increasing. 17  
every 17  
existing 13  
few 17  
for 85  
framework 17  
free 6  
from 17  
functional 17  
generally 17  
gets 17  
given 17  
global 17  
good 17  
handling, 17  
has 34  
highest-level 17  
hold 17  
how 17  
hundreds 17  
implicit 17  
impossible 34  
in 64  
increasing 17  
into 13  
is 238  
it 51  
just 17  
known 17  
language 17  
large 51  
less 17  
license. 17  
licensed 17  
lies 17  
lock 6  
manage 17  
management 17  
named 17  
news 17  
next 17  
not 17  
nothing 17  
obsolete. 17  
of 197  
on 77  
on-hand 17  
open 17  
optimize 13  
or 17  
originally 17  
overlook 17  
panacea 17  
paper 17  
petabytes 17  
problem 17  
process 34  
process, 17  
processing 17  
programming 17  
programming. 17  
project 17  
retrieve 17  
retrieving 17  
search 17  
see 17  
sets 17  
share, 17  
simpler 17  
since 17  
site 13  
so 17  
software 17  
son�s 17  
soon 17  
source 17  
standard 17  
stop 17  
store 17  
store, 34  
stores. 17  
storing, 17  
structure, 6  
success 17  
success. 17  
supports 17  
system 17  
systems 34  
terabytes, 17  
term 17  
terms 17  
than 17  
that 68  
the 153  
them 17  
therefore 17  
these 34  
this, 17  
this�Hadoop, 17  
those 17  
to 132  
tools 17  
toy 17  
traditional 34  
under 17  
use 17  
used 17  
using 17  
v2 17  
variety 17  
various 17  
volumes 17  
vs 6  
was 17  
were 17  
which 34  
with 48  
working 17  
world 17  
written 34  
years 17  
yellow 17  
you 17  
�Big 17

Path of input file on VMware server is :- /wordcount/input/wordcountproblem.txt

Path of the output file on VMware server is :- /wordcount/output/part-00000

Question 2)

--------------

Following commands are executed in following order :-

i.  hadoop fs copyFromLocal wordcount.java /user/JigBD0008/wordcount

ii. hadoop fs -cat /user/JigBD0008/wordcount/wordcount.java

output of above command will produce below result:-

package com.jigsaw;  
import java.io.IOException;  
import java.util.\*;

/\* import all hadoop related libraries \*/  
import org.apache.hadoop.conf.\*;  
import org.apache.hadoop.fs.\*;  
import org.apache.hadoop.io.\*;  
import org.apache.hadoop.mapreduce.\*;  
import org.apache.hadoop.mapreduce.lib.input.\*;  
import org.apache.hadoop.mapreduce.lib.output.\*;  
import org.apache.hadoop.util.\*;

/\* Configurable is the interface and Configured implements it.   
Configured then becomes the base class that implements setConf() and getConf() classes of Configurable interface.  
WordCount class extends Configured and hence it gets configured using a Configuration class.

Motive - TOol interface is implemented and Configured is inherited becuase we can easily set Configuration object via GenericOptionsParser. That means we can pass any command line arg on run time, apart from i/p file and o/p file.\*/

public class WordCount extends Configured implements Tool {

/\* Mapper is base class here that takes i/p as LongWritable(key) and Text (value) and give o/p as Text(key) and IntWritable(value)\*/  
public static class WCMapper extends Mapper<LongWritable, Text, Text, IntWritable> {  
String line; /\* A string reference\*/  
StringTokenizer tokenizer; /\* It converts a string into tokens.\*/  
/\* o/p of map value will be written into this object. It has been initialized with 1, using constructor. If it is intialized with '0', or NULL (blank) then count of the words will come as 0. It is because the object of IntWritable would be initialized with '0'.\*/  
private final static IntWritable one = new IntWritable(1);  
  
/\* o/p of map key would be written using this object\*/  
private Text word = new Text();

/\* map is the main function() into which we implement our code. Here, it will take LongWritable as key and Text as value. Context is passed through out the program, such as inside map, setup, and cleanup method. We can set any variable in setup() method, and use it inside map() method using Context.\*/   
public void map(LongWritable key, Text value, Context context)   
throws IOException, InterruptedException {  
/\* Convert a i/p (here line) into string format. \*/  
line = value.toString();  
/\* converts a line into words(tokens). By default it tokenizes using blank spaces.\*/  
tokenizer = new StringTokenizer(line);  
/\* Using tokens, we write them into context that will be availbel as i/p to reducer.\*/  
while (tokenizer.hasMoreTokens()) {  
word.set(tokenizer.nextToken());  
/\* context is used to communicate to mapreduce framework. Hence writing here will be available to reducer.\*/  
context.write(word, one);  
}  
}  
}  
/\* WcReducer is the class that extends Reducer, which takes i/p in the form of Text(key) and IntWritable(value). Same as i/p, it set o/p as Text(key) and IntWritable(value) \*/   
public static class WCReducer extends Reducer<Text, IntWritable, Text, IntWritable> {

/\* reduce() is the method in which we write our code. It accepts i/p as in the formof Text(key) and many values for a key, hence the vale is passed using Iterable. O/pO/p will be written using Cotext so that the o/p will be acceptable throughout mapreduce program.\*/  
  
@Override public void reduce(Text key, Iterable<IntWritable> val, Context context) throws IOException, InterruptedException {  
int sum = 0;  
/\* Iterating over the values \*/  
Iterator<IntWritable> values = val.iterator();  
/\* taking sum of all the values for a key\*/  
while (values.hasNext()) {  
sum += values.next().get();  
}  
/\* writing the sum of a key ( word) into context.\*/  
context.write(key, new IntWritable(sum));  
}  
}  
  
/\* this Tool.run(String[]) is executed by run(Configuration, Tool, String[]) after parsing with the given generic args. Uses the given Configuration or builds a new obj if it is null. If any new arg is passed using command line, then it would override the Configuration. Finally this method executes the command with the given arguments\*/  
public int run(String[] args) throws Exception {  
  
/\*this Configuration is processed by ToolRunner. A new Configuration object is created that will have access to the hadoop cluster's configuration that is set using the core-site.xml, mapred.xml, and hdfs-site.xml files.\*/  
Configuration conf = new Configuration();

/\*created a new job using processed conf\*/  
Job job = new Job(conf);

/\*job specific parameter is set using class. On command line along with option 'jar' the class is passed and that class is passed here. It is used to set the jar by finding where a given class came from.\*/  
job.setJarByClass(WordCount.class);

/\*InputFormat interface is responsible for the i/p specification for a map-red job. map-red framework depends on the InPutFormat of the job to validate i/p specification of the job, split the i/p files into InputSplits (each of which is then assigned to an individual Mapper), and provide an implementation of RecordReader, which is responsible to respect the record-boundaries and present a record-oriented view of the logical InputSplit to the individual task. Generally hadoop framework spaws one map task for each InputSplit generated by InputFormat.  
Here, TextInputFormat is the InputFormat for the plain text files. Files are broken into lines and either linefeed or carriage-return is the delimiter for EOL. Keys are the position in the file and values are the lines of the text.  
RecordReader is also responsible for generating <key,value> pair for the input split generated by InputFormat objects.  
\*/  
job.setInputFormatClass(TextInputFormat.class);

/\* Map-reduce will accept the input from the path set on command-line after classname. More than 1 path can be added using comma separated paths.\*/  
TextInputFormat.addInputPath(job, new Path(args[0]));

/\*sets the mapper for the job\*/  
job.setMapperClass(WCMapper.class);

/\*sets the reducer for the job.\*/  
job.setReducerClass(WCReducer.class);

/\*we can tell mar-reduce job how many reducers are needed. We can also pass that value from generic command-line argument. For each reducer, there will be job created in the output path given at command-lne. The format of the reducer output will be like part-00000, part-00001, etc. If the data was sufficinet to get processed from one reducer, then the output will come only in say part-00000. part-00001 will be blank in that case.\*/   
job.setNumReduceTasks(2);

/\*OutputFormat depicts the output-specification for a map-red job. map-red framework relies on the OutputFormat for the job to validate the output-specification of the job (like checking whether o/p directory already exists), and provide RecordWriter implementation to write the output files of the job. Output files are stored in filesystem.   
Here, the output will be plain text.\*/  
job.setOutputFormatClass(TextOutputFormat.class);  
  
/\*set the key class for the job output data. Here it will be plain text.\*/  
job.setOutputKeyClass(Text.class);

/\*set the value class for the job output data. Here it will be IntWritable.\*/  
job.setOutputValueClass(IntWritable.class);

/\*set the key class for map output data. It provides flexibility to users to any class that may be different that the final output key class.\*/  
job.setMapOutputKeyClass(Text.class);

/\*set the value class for map output data. It provides flexibility to users to any class that may be different that the final output value class. It may be say FloatWritable if we have to parse Float values.\*/  
job.setMapOutputValueClass(IntWritable.class);  
  
/\*path of the output directory for the map-red job is set, which is set as last argument of the comman-line argument while running a hadoop job.\*/  
TextOutputFormat.setOutputPath(job, new Path(args[1]));

/\*submit the job to the cluster and wait for the job to finish. \*/  
boolean res = job.waitForCompletion(true);  
if (res)  
return 0;  
else  
return -1;   
}

public static void main(String args[]) throws Exception {

/\*ToolRunner is used to handle generic command line options\*/  
int res = ToolRunner.run(new WordCount(), args);  
System.exit(res);  
}  
}