과제2-1

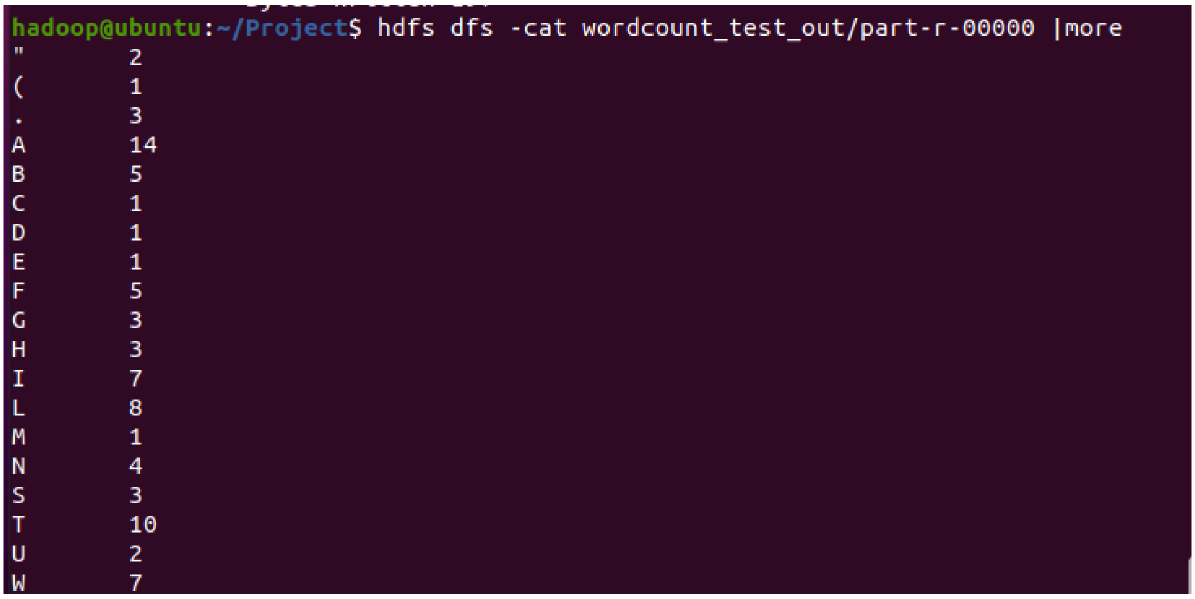
목표

1. wordcountsort 분산처리를 해보자

간단설명

알파벳순으로 모은 wordcount를 partition을 이용하여 알파벳순으로 정리하였다

결과



```
hadoop@ubuntu:~/Project$ hdfs dfs -cat wordcount_test_out/part-r-00000 |more
"       2
(       1
.       3
A       14
B       5
C       1
D       1
E       1
F       5
G       3
H       3
I       7
L       8
M       1
N       4
S       3
T       10
U       2
W       7
```

code

```java
package ssafy;

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;
import org.apache.hadoop.util.GenericOptionsParser;
import org.apache.hadoop.mapreduce.Partitioner;
```

```
public class Wordcountsort {
    /*
    Object, Text : input key-value pair type (always same (to get a line of input
file))
    Text, IntWritable : output key-value pair type
    */
    public static class MyPartitioner extends           Partitioner<Text,
IntWritable> {
    @Override
    public int getPartition(Text key, IntWritable value, int numPartitations){
    if (key.toString().charAt(0)<'a') return 0;
    else return 1;
    }
}
    public static class TokenizerMapper
            extends Mapper<Object,Text,Text,IntWritable> {

        // variable declairations
        private final static IntWritable one = new IntWritable(1);
        private Text word = new Text();

        // map function (Context -> fixed parameter)
        public void map(Object key, Text value, Context context)
                throws IOException, InterruptedException {

            // value.toString() : get a line
            StringTokenizer itr = new StringTokenizer(value.toString());
            while ( itr.hasMoreTokens() ) {
                word.set(itr.nextToken().substring(0,1));

                // emit a key-value pair
                context.write(word,one);
            }
        }
    }

    /*
    Text, IntWritable : input key type and the value type of input value list
    Text, IntWritable : output key-value pair type
    */
    public static class IntSumReducer
            extends Reducer<Text,IntWritable,Text,IntWritable> {

        // variables
        private IntWritable result = new IntWritable();

        // key : a disticnt word
        // values :  Iterable type (data list)
        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);
```

```java
            context.write(key,result);
        }
    }


    /* Main function */
    public static void main(String[] args) throws Exception {
        Configuration conf = new Configuration();
        String[] otherArgs = new
GenericOptionsParser(conf,args).getRemainingArgs();
        if ( otherArgs.length != 2 ) {
            System.err.println("Usage: <in> <out>");
            System.exit(2);
        }
        Job job = new Job(conf,"word count");
        job.setJarByClass(Wordcountsort.class);

        // let hadoop know my map and reduce classes
        job.setMapperClass(TokenizerMapper.class);
        job.setReducerClass(IntSumReducer.class);

        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);
        job.setPartitionerClass(MyPartitioner.class);

        // set number of reduces
        job.setNumReduceTasks(2);

        // set input and output directories
        FileInputFormat.addInputPath(job,new Path(otherArgs[0]));
        FileOutputFormat.setOutputPath(job,new Path(otherArgs[1]));
        System.exit(job.waitForCompletion(true) ? 0 : 1 );
    }
}
```