1번 문제.

HADOOP 에서 HDFS에 읽었었는데,MONGODB파일을 HADOOP에서 불러오자.

Start a MongoDB container

> docker run -d --name mongodb -p 27017:27017 mongo

Explain what does the -p option does? Is the option mandatory to run instructions in the next slides?

In the installed MongoDB container, load movies.csv, tags.csv

이게 무엇인가? 이 OPTION이 왜 필요한것인가

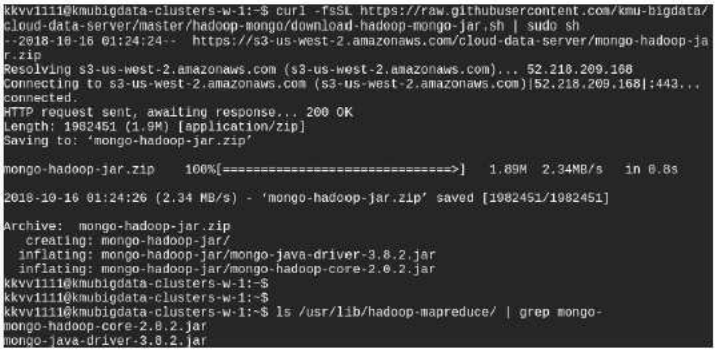
1번 ANSWSER

DOCKER –P OPTION은 컨테이너 안에서 서비스를 구동하고, 외부에서 그 서비스에 접근하려고 할 때 사용한다. IP와 PORT로 그 컨테이너 안 서비스에 접속한다. 여기서는 27017:27017 中 첫 번째는 연결할 호스트의 포트 번호 이고, 뒤는 컨테이너의 포트 번호이다. 즉, -P 옵션은 컨테이너 포트를 열고 호스트 포트와 연결하는 옵션이다. 따라서 이 옵션을 사용하지 않았더라면, 뒤의 슬라이드를 진행하는데 어려움이 있다. 왜냐하면 MASTER NODE에서 컨테이너를 연결하고 실습을 진행하는 내용이기 때문이다.

Download mongo-hadoop library from slave nodes

$ curl -fsSL \
https://raw.githubusercontent.com/kmu-bigdata/cloud-data-server/master/hadoop-mongo/download-hadoop-mongo-jar.sh | sudo sh
$ ls /usr/lib/hadoop-mapreduce/ | grep mongo-

Explain why the steps in page 10~12 are necessary.



2번 ANSWER

10~12P의 내용들은 HOST(LOCAL)에서 MONGOBD컨테이너에파일을 COPY하고 컨테이너에서는 MONGODB로 파일을 옮긴다. 그 후, 3개의 WORKER NODE에서 MONGOIMPORT를 하는데 여기서는 MONGODB에 데이터를 넣는 것이다. 후자의 경우는 WORKER NODE 3개에 걸쳐 모두 진행한다. 컨테이너의 경우는 모든 NODE에서 접근할 수 있지만 MONGO에서 처리할 데이터의 경우는 각 NODE마다 줘야 하기 때문이다.

MovieIdByTags Code

Explain what this code does

코드설

```java
import com.mongodb.hadoop.MongoInputFormat;
import com.mongodb.hadoop.MongoOutputFormat;
import com.mongodb.hadoop.io.BSONWritable;
import com.mongodb.hadoop.util.MongoConfigUtil;

public class MovieIdByTags {
    public static void main(String[] args) throws Exception {
        Configuration conf = new Configuration();

        MongoConfigUtil.setInputURI(conf, "mongodb://" + args[0]);
        MongoConfigUtil.setOutputURI(conf, "mongodb://" + args[1]);

        Job job = Job.getInstance(conf, "TagsAggregator");

        job.setJarByClass(MovieIdByTags.class);

        job.setMapperClass(Map.class);
        job.setCombinerClass(Combine.class);
        job.setReducerClass(Reduce.class);

        job.setMapOutputKeyClass(Text.class);
        job.setMapOutputValueClass(Text.class);

        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(BSONWritable.class);

        job.setInputFormatClass(MongoInputFormat.class);
        job.setOutputFormatClass(MongoOutputFormat.class);

        job.waitForCompletion(true);
    }
```

CAPTURE ANSWER

DRIVER CODE로서 각 단계의 입출력 형태를 지정한다. 이외에 기본적인 CONFIGURATION한다.

➔ MONGODB와 HADOOP간의 DATA를 주고 받기 위해서 CONFIGURATION을 설정하고 관련 API를 IMPORT한다. MR 작업을 위해 JOB 한 개를 생성한다. (WORDCOUNT의 경우는 JOB2개를 만들었다. 마지막에 SORT를 위한 MR작업을 위해) 그 후, MOVIEIDBYTAGS클래스를 바탕으로 하는 JAR파일을 만들도록 설정한다. 이후, MAPPER, COMBINER, REDUCE작업에 대한 클래스를 연결한다. 또한 각 단계별 INPUT, OUTPUT FORMAT을 설정한다.

전체 MAP의 INPUT, OUTPUT TYPE은 TEXT로 설정한다.

SETMAPOUTPUTKEYCLASS와 SETMAPOUTPUTVALUECLASS에서는 MAP의 출력에 대한 것인데, TEXT 형태로 KEY, VALUE 를 RETURN 할 것을 말한다. 원래는 SETOUTPUTKEYCLASS 등을 이용해서 한 번에 작성하지만, MAP과 REUDCE의 TYPE이 다르기 때문에 따로 설정한다.

또한 SETOUTPUTKEYCLASS, SETOUTPUTKEYVALUECLASS를 통해서는 최종 결과물에 대해서 TYPE을 정하는데 KEY, VALUE를 KEY는 TEXT이고 VALUE는 BSONWRITABLE로 이진 JSON TYPE으로, SERIALIZATION가능한 TYPE으로 보낸다는 것이다.

마지막으로 SETINPUTFORMATCLASS, SETOUTPUTFORMATCLASS를 정의하는데 이는, 전체 입출력 DATA FORMAT을 MONGO_FORMAT으로 설정한다는 의미다.

MovieIdByTags Code

MovieIdByTags Map

Explain what this code does

```
public static class Map extends Mapper<ObjectId, BSONObject, Text, Text> {
    private final Text tagOutput = new Text();
    private final Text movieIdOutput = new Text();

    public void map(ObjectId key, BSONObject value, Context context)
                throws IOException, InterruptedException {
        String tag = value.get("tag").toString();
        String movieId = value.get("movieId").toString();

        tagOutput.set(tag);
        movieIdOutput.set(movieId);
        context.write(tagOutput, movieIdOutput);
    }
}
```

CAPTURE ANSWER

→MAP에서는 MAP의 INPUT KEY, VALUE를 OBJECTID와 BSONOBJECT(BINARY JSON)TYPE으로 정하고 OUTPUT의 KEY, VALUE는 TEXT TYPE으로 정한다. 또한 변수를 설정하는데 TAGOUTPUT, MovieIdOutput 은 text type의 상수형태다. MAP으로 들어오는 KEY-VALUE 입력에 대해서 TAG는 VALUE에서 TAG를 가져온 값을 STRING형태로 저장한다. 또한 MOVIEID에 MOVIEID를 가져와서 STRING형태로 가져온다. 그리고 TAG값을 tagoutput변수에 저장한다. 도한 movieIdOutput변수에 movieId값을 저장한다. 그 후에 tagsoutput과 movieIdOutput을 저장한다. 즉, map의 결과로 return한다. 정리하자면 value로 오는 BSONObjet에 대해서 tag와 mvoidId값을 각각 tagoutput과 movieidoutput으로 저장한 후 key는 tagoutput value는 ovieIdOutput을 저장하여 reutrn한다. 이때 둘의 type은 text type이다.

MovieIdByTags Code

Explain what Combiner does

```
public static class Combine extends Reducer<Text, Text, Text, Text> {
    private final Text combinerOutput = new Text();

    public void reduce(Text key, Iterable<Text> values, Context context)
                    throws IOException, InterruptedException {
        HashSet<String> ids = new HashSet<>();
        for (Text value : values) {
            ids.add(value.toString());
        }

        Iterator itr = ids.iterator();
        while (itr.hasNext()) {
            combinerOutput.set(itr.next().toString());
            context.write(key, combinerOutput);
        }
    }
}
```

Capture Answer

➔먼저, combine이 MR PROGRAMMING에 있어서 어떤 역할을 하는지 설명하겠다. 이는 Optional로서 Map의 결과를 Local Write하고 Reduce에 보내는 과정에 있어서 그 때 각 map의 결과를 합치는 것이다. 예를 들어서 고양이1, 고양이1, 강아지1 이라고 map의 결과가 나온다면, 고양이2 강아지1로 Map의 결과물들을 combine한다.

여기서의 Combine을 설명하자면, Input, Output의 key-value가 모두 Text type이다. 입력에 대해서 설명하자면 TYPE은 TEXT이지만, VALUE는 TEXT ITERABLE이다. 즉, KEY별로 VALUE를 ITERABLE하게 가지고 있다. 이것에 대해 TRAVERSE하는데, KEY마다 VALUE를 HASH자료구조에 STRING으로 바꾸어 저장한다. 또한 HASHSET에 대해서 ITERATOR를 이용해서 COMBINEROUTPUT변수에 HASH에 저장된 값을 저장한다. 그리고 해당 KEY(TAG)와 결과를VALUE(MOVIEID)로 RETURN한다.

MovieIdByTags Code

MovieIdByTags Reduce

Explain what this code does

```
public static class Reduce extends Reducer<Text, Text, Text, BSONWritable> {
    private BSONWritable reduceResult = new BSONWritable();

    public void reduce(Text key, Iterable<Text> values, Context context)
                    throws IOException, InterruptedException {
        BasicBSONObject output = new BasicBSONObject();
        HashSet<String> ids = new HashSet<>();
        StringBuilder sb = new StringBuilder();

        for (Text value : values) {
            ids.add(value.toString());
        }

        Iterator itr = ids.iterator();
        while (itr.hasNext()) {
            sb.append(itr.next().toString());
            sb.append(", ");
        }

        String movieIds = sb.toString();
        if (movieIds.endsWith(", ")) {
            movieIds = movieIds.substring(0, movieIds.length() - 2);
        }

        output.put("movieIds", movieIds);
        reduceResult.setDoc(output);
        context.write(key, reduceResult);
    }
  }
}
```

CAPTURE ANSWER

→해당 코드는 REDUCE 과정이다. REDUCE의 INPUT가 OUTPUT KEY-VALUE는 INPUT의 경우는 COMBINE과정을 거친 것으로 KEY와 VALUE는 TEXT이다. 또한 OUTPUT의 KEY는 TEXT이고 VALUE는 BSONWRITABLE TYPE으로써 BINARY JSON TYPE이다. KEY(TAG)마다 가진 MOVIEID(ITERABLE MOVIEID)를 읽어서 HASHSET에 저장한다. (이 때, SET이므로 중복이 없다.) 그리고 HASHSET을 ITERATOR로 TRAVERSE하면서 STRINGBUILDER TYPE가진 SB변수에 STRING TYPE으로 MOVIEID를 APPEND해서 저장한다. 또한 MOVIEIDS라는 STRING에 MOVIEID를 APPEND해서 저장한 SB를 저장한다. 그 후, 해당 STRING TYPE의 MOVIEIDS가 "," 형태로 끝난다면 그 앞까지 잘라서 MOVIEIDS에 저장한다. 그 후, BASICBSONOBJECT에 "MOVIEIDS"이름으로 앞의 과정 결과를 저장한다. 그 후 BSONWRITABLE TYPE형식에 맞게 SETDOC함수를 통해 저장하고 REDUCE를 마친다.

결과, TAG별 MOVIEID 모음이 저장된다. (BSON TYPE을 MONGODB로 읽어서 출력해볼 수 있다)

Capture the screen of MongoDB outputs of

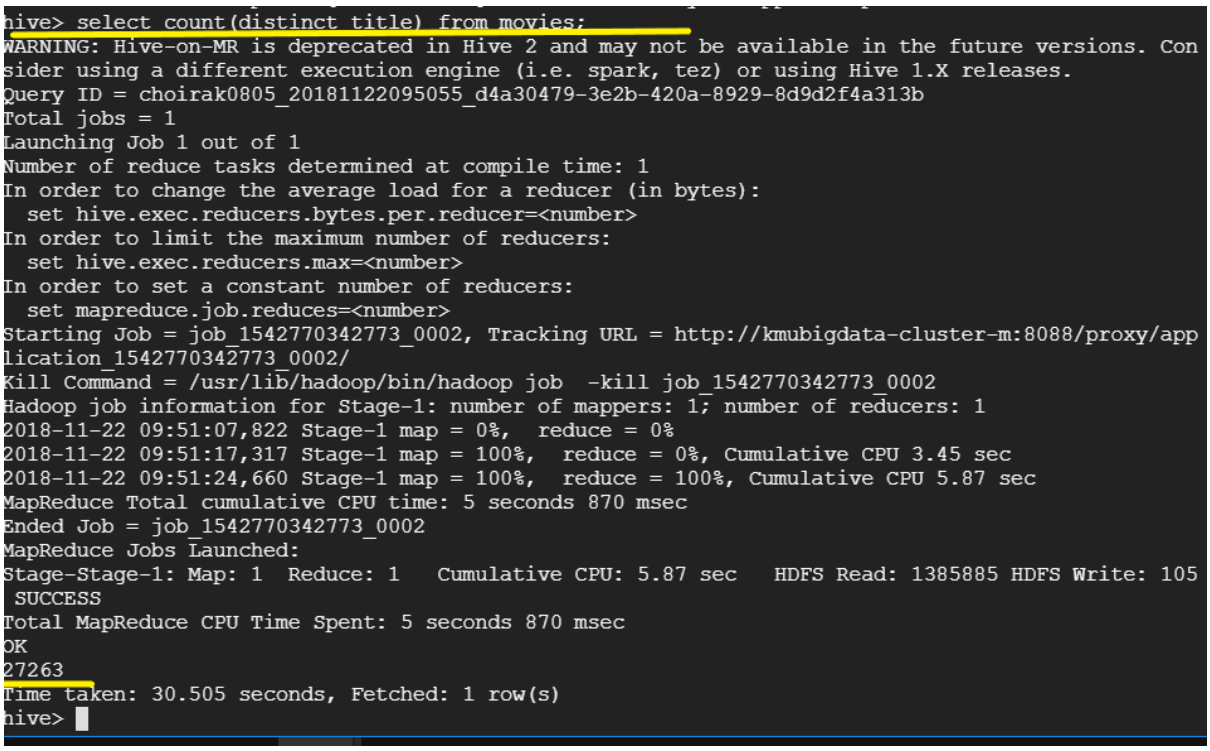
movielens.tags_out table

-capture결과- -

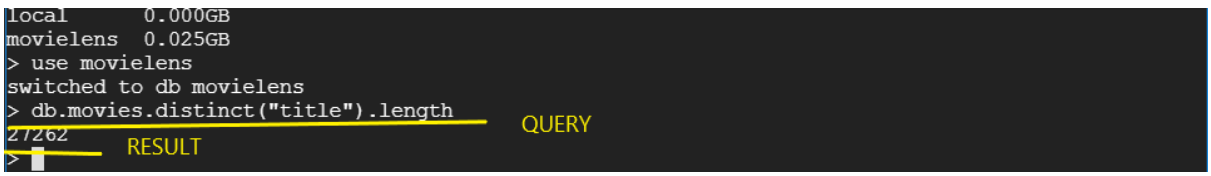
==EXERCISE==

1. ***Using the movielens dataset, write a MongoDB query and Hive query to count the number of unique movie names (movies.csv)***

-HIVE-



-MongoDB-



2. Implement Hadoop MapReduce code to count the number of unique movie names

DRIVER

```
import java.io.IOException;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
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.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;
import java.util.Arrays;
import java.util.ArrayList;

public class COUNT_UNIQUE_DRIVER {

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

                Configuration conf=new Configuration();


                Job job = Job.getInstance(conf,"COUNT_UNIQUE_DRIVER");

                job.setJarByClass(COUNT_UNIQUE_DRIVER.class);

                job.setNumReduceTasks(1);//To get unique title from multiple results from mapper

                job.setOutputKeyClass(Text.class);//KEY-VALUE (MOVIE_TITLE - COUNT)
                job.setOutputValueClass(IntWritable.class);

                job.setMapperClass(COUNT_UNIQUE_MAP.class);
                job.setReducerClass(COUNT_UNIQUE_REDUCE.class);

                job.setInputFormatClass(TextInputFormat.class);
                job.setOutputFormatClass(TextOutputFormat.class);

                FileInputFormat.addInputPath(job,new Path(args[0]));
                FileOutputFormat.setOutputPath(job,new Path(args[1]));

                job.waitForCompletion(true);

        }
```

MAP

```
        }

public static class COUNT_UNIQUE_MAP extends Mapper<LongWritable,Text,Text,IntWritable> {

        private final static IntWritable ONE=new IntWritable(1);
        private Text word=new Text();
        ArrayList<String> titles=new ArrayList<String>();


        public void map(LongWritable key,Text value,Context context)throws IOException,InterruptedException{
                String line=value.toString();//READ LINE-BY-LINE
                String [] fields=line.split(",");//DECODE -CSV
                String title=fields[fields.length-2];//GET TITLE
                //String[] titles=want_field;


                boolean whether=true;

                if(titles.contains(title)){
                        whether=false;
                }

                //Compare whether this title is saved in Word
                if (whether==true) {
                        titles.add(title);
                        word.set(title);
                        context.write(word,ONE);//Return KEY-VALUE
                }
        }
}




public static class COUNT_UNIQUE_REDUCE extends Reducer<Text,IntWritable,Text,IntWritable> {

        /*
         * REDUCE RETURN UNIQUE TITLE-ONE PER BLOCK. SO WE SHOULD GET UNIQUE IN REDUCE
         */
```

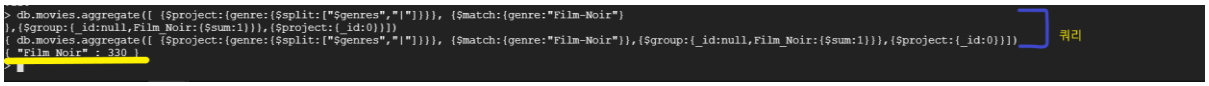
REDUCE

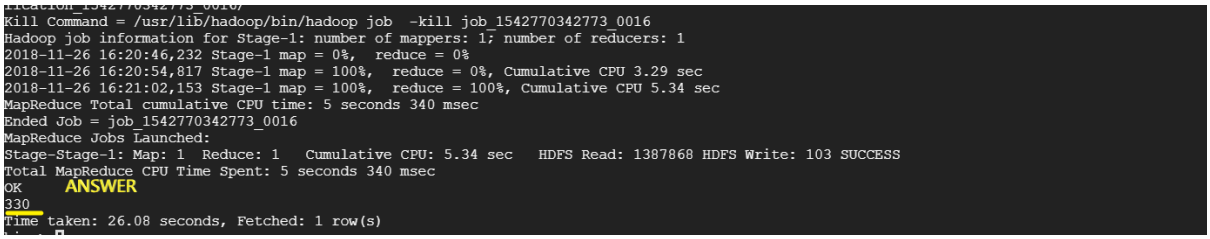
```
        * REDUCE RETURN UNIQUE TITLE-ONE PER BLOCK. SO WE SHOULD GET UNIQUE IN REDUCE
        */

        int for_res=0;
        Text for_key= new Text();


        public void reduce(Text key,Iterable<IntWritable> values,Context context) throws IOException,InterruptedException{

                int sum=0;



                for (IntWritable val:values) {

                        sum+=val.get();

                }
                if (sum==1) {
                        //context.write(key,new IntWritable (sum));

                        for_res+=1;
                }
        }

        protected void cleanup(Context context) throws IOException, InterruptedException{
                for_key.set(" ");
                context.write(for_key,new IntWritable(for_res));
        }

}

}
```

```
        HDFS: Number of write operations=3
    Job Counters
        Launched map tasks=1
        Launched reduce tasks=1
        Rack-local map tasks=1
        Total time spent by all maps in occupied slots (ms)=26576
        Total time spent by all reduces in occupied slots (ms)=13808
        Total time spent by all map tasks (ms)=6644
        Total time spent by all reduce tasks (ms)=3452
        Total vcore-milliseconds taken by all map tasks=6644
        Total vcore-milliseconds taken by all reduce tasks=3452
        Total megabyte-milliseconds taken by all map tasks=13606912
        Total megabyte-milliseconds taken by all reduce tasks=7069696
    Map-Reduce Framework
        Map input records=27279
        Map output records=27263
        Map output bytes=891651
        Map output materialized bytes=946196
        Input split bytes=112
        Combine input records=0
        Combine output records=0
        Reduce input groups=27263
        Reduce shuffle bytes=946196
        Reduce input records=27263
        Reduce output records=1
        Spilled Records=54526
        Shuffled Maps =1
        Failed Shuffles=0
        Merged Map outputs=1
        GC time elapsed (ms)=175
        CPU time spent (ms)=5520
        Physical memory (bytes) snapshot=717983744
        Virtual memory (bytes) snapshot=6973521920
        Total committed heap usage (bytes)=643301376
    Shuffle Errors
        BAD_ID=0
        CONNECTION=0
        IO_ERROR=0
        WRONG_LENGTH=0
        WRONG_MAP=0
        WRONG_REDUCE=0
    File Input Format Counters
        Bytes Read=1377677
    File Output Format Counters
        Bytes Written=8
choirak0805@kmubigdata-cluster-m:~/HW_03_MR/Q02$ hdfs dfs -cat /hw03/Q02/final_output/*
18/11/25 01:58:14 INFO gcs.GoogleHadoopFileSystemBase: GHFS version: 1.6.10-hadoop2
    27263
<--RESULT
choirak0805@kmubigdata-cluster-m:~/HW_03_MR/Q02$
```

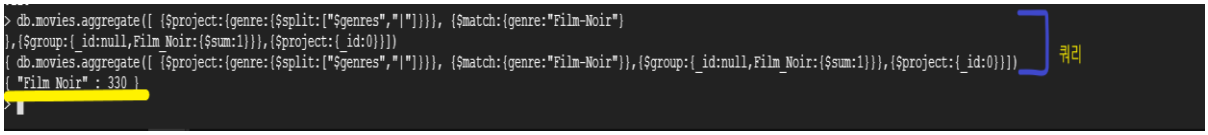
3. Using the movielens dataset, write a MongoDB query and Hive query to Count the number of movies whose genre contains a word "Film-Noir" (movies.csv)

-HIVE-





-MongoDB—



4. Implement Hadoop MapReduce code to count the number of movies whose genre contains a word "Film-Noir" (movies.csv)

DRIVER

```
import java.io.IOException;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
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.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;
import java.util.Arrays;
import java.util.ArrayList;

public class COUNT_FILM_NOIR {

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

                Configuration conf=new Configuration();


                Job job = Job.getInstance(conf,"COUNT_FILM_NOIR");

                job.setJarByClass(COUNT_FILM_NOIR.class);

                job.setNumReduceTasks(1);//To SUM "FILM-NOIR" GENRE TOGETHER AT ONE-REUDCE

                job.setOutputKeyClass(Text.class);//KEY-VALUE (MOVIE_TITLE - COUNT)
                job.setOutputValueClass(IntWritable.class);

                job.setMapperClass(COUNT_FILM_NOIR_MAP.class);
                job.setReducerClass(COUNT_FILM_NOIR_REDUCE.class);

                job.setInputFormatClass(TextInputFormat.class);
                job.setOutputFormatClass(TextOutputFormat.class);

                FileInputFormat.addInputPath(job,new Path(args[0]));
                FileOutputFormat.setOutputPath(job,new Path(args[1]));

                job.waitForCompletion(true);
```

```
                job.waitForCompletion(true);

        }

public static class COUNT_FILM_NOIR_MAP extends Mapper<LongWritable,Text,Text,IntWritable> {

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



        public void map(LongWritable key,Text value,Context context)throws IOException,InterruptedException{
                String line=value.toString();//READ LINE-BY-LINE
                String [] fields=line.split(",");//DECODE -CSV
                String lastField=fields[fields.length-1];//GET GENRE
                String [] genres = lastField.split("\\|");
                //ONLY pass "FILM-NOIR"
                for (String genre:genres) {
                        if(genre.equals("Film-Noir")) {
                                word.set(genre);
                                context.write(word,ONE);
                        }
                }

        }


}



public static class COUNT_FILM_NOIR_REDUCE extends Reducer<Text,IntWritable,Text,IntWritable> {

        /*
         * REDUCE RETURN UNIQUE TITLE-ONE PER BLOCK. SO WE SHOULD GET UNIQUE IN REDUCE
         */
        Text for_key=new Text();
        int sum=0;

        public void reduce(Text key,Iterable<IntWritable> values,Context context) throws IOException,InterruptedException{
```

MAP

GNU nano 2.7.4 File: COUNT_FILM_NOIR.java

```
    public void reduce(Text key,Iterable<IntWritable> values,Context context) throws IOException,InterruptedException{


            for_key.set(key);
            for (IntWritable val:values) {

                    sum+=val.get();
            }

    }
            protected void cleanup(Context context ) throws IOException ,InterruptedException{

                    context.write(for_key,new IntWritable(sum));


            //context.write(key,new IntWritable (sum));
    }


    }
}
```

REDUCE

[Read 107 lines]

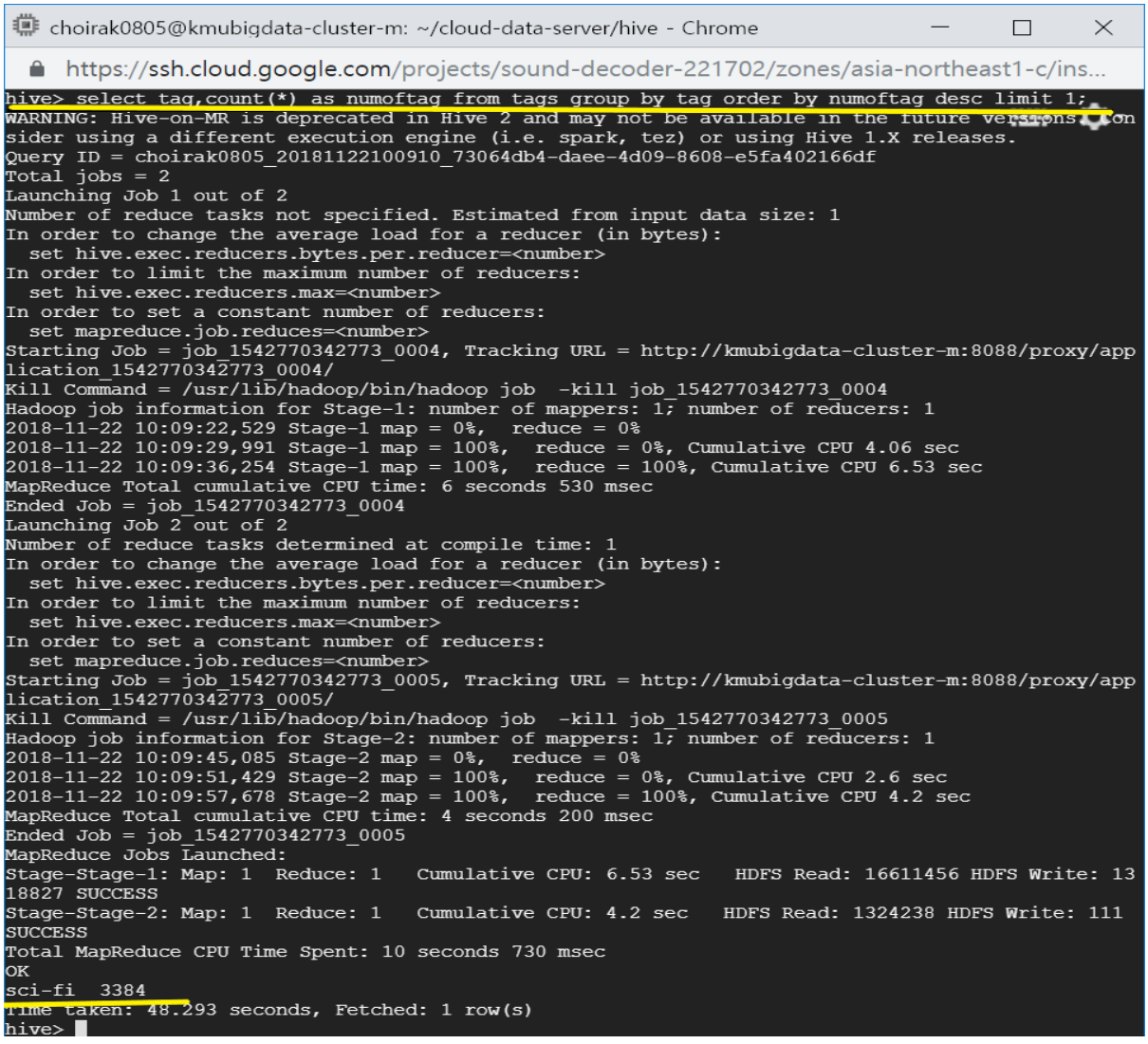
^G Get Help ^O Write Out ^W Where Is ^K Cut Text ^J Justify ^C Cur Pos ^Y Prev Page M-\ First Line M-W WhereIs Next ^^ Mark Text M-} Indent Text
^X Exit ^R Read File ^\ Replace ^U Uncut Text ^T To Spell ^_ Go To Line ^V Next Page M-/ Last Line M-] To Bracket M-^ Copy Text M-{ Unindent Text

```
        HDFS: Number of write operations=3
    Job Counters
        Launched map tasks=1
        Launched reduce tasks=1
        Data-local map tasks=1
        Total time spent by all maps in occupied slots (ms)=16676
        Total time spent by all reduces in occupied slots (ms)=14208
        Total time spent by all map tasks (ms)=4169
        Total time spent by all reduce tasks (ms)=3552
        Total vcore-milliseconds taken by all map tasks=4169
        Total vcore-milliseconds taken by all reduce tasks=3552
        Total megabyte-milliseconds taken by all map tasks=8538112
        Total megabyte-milliseconds taken by all reduce tasks=7274496
    Map-Reduce Framework
        Map input records=27279
        Map output records=330
        Map output bytes=4620
        Map output materialized bytes=5286
        Input split bytes=112
        Combine input records=0
        Combine output records=0
        Reduce input groups=1
        Reduce shuffle bytes=5286
        Reduce input records=330
        Reduce output records=1
        Spilled Records=660
        Shuffled Maps =1
        Failed Shuffles=0
        Merged Map outputs=1
        GC time elapsed (ms)=172
        CPU time spent (ms)=1870
        Physical memory (bytes) snapshot=721809408
        Virtual memory (bytes) snapshot=6972743680
        Total committed heap usage (bytes)=642777088
    Shuffle Errors
        BAD_ID=0
        CONNECTION=0
        IO_ERROR=0
        WRONG_LENGTH=0
        WRONG_MAP=0
        WRONG_REDUCE=0
    File Input Format Counters
        Bytes Read=1377677
    File Output Format Counters
        Bytes Written=14
choirak0805@kmubigdata-cluster-m:~/HW_03_MR/Q04$ hdfs dfs -cat /hw03/Q04/final_final_output/*
18/11/25 01:47:45 INFO gcs.GoogleHadoopFileSystemBase: GHFS version: 1.6.10-hadoop2
Film-Noir       330
choirak0805@kmubigdata-cluster-m:~/HW_03_MR/Q04$
```

<-RESULT

5. Using the movielens dataset, write a MongoDB query and Hive query to list the tag that are cited the most (tags.csv)

-HIVE-



-MongoDB--


{ "Film_Noir" : 14 }
> db.tags.aggregate([
... {$group:{_id:"$tag",numoftag:{$sum:1}}},
... {$sort:{numoftag:-1}},
... {$limit:1}])
{ "_id" : "sci-fi", "numoftag" : 3384 }
> 20142770 ChoiRakJun To Certification

6. Implement Hadoop MapReduce code to list the tag that are cited the most (tags.csv)

DRIVER

```
import java.io.IOException;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
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.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;
import java.util.Arrays;
import java.util.ArrayList;

public class COUNT_MOST_CITED {

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

                Configuration conf = new Configuration();

                Job job = Job.getInstance(conf, "COUNT_MOST_CITED");

                job.setJarByClass(COUNT_MOST_CITED.class);
                job.setNumReduceTasks(1);//TO get unique title from multiple results from mapper

                // job.setNumReduceTasks(1);//To get unique title from multiple results from
                // mapper

                job.setOutputKeyClass(Text.class);// KEY-VALUE (MOVIE_TITLE - COUNT)
                job.setOutputValueClass(IntWritable.class);

                job.setMapperClass(COUNT_MOST_CITED_MAP.class);
                job.setReducerClass(COUNT_MOST_CITED_REDUCE.class);

                job.setInputFormatClass(TextInputFormat.class);
                job.setOutputFormatClass(TextOutputFormat.class);

                FileInputFormat.addInputPath(job, new Path(args[0]));
                FileOutputFormat.setOutputPath(job, new Path(args[1]));

                job.waitForCompletion(true);
```

```
        job.waitForCompletion(true);

    }

    public static class COUNT_MOST_CITED_MAP extends Mapper<LongWritable, Text, Text, IntWritable> {

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

        public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {
            String line = value.toString();// READ LINE-BY-LINE
            String[] fields = line.split(",");// DECODE -CSV
            String tag = fields[fields.length - 2];// GET TAG
            // String[] titles=want_field;

            word.set(tag);
            context.write(word, ONE);

        }

    }

    public static class COUNT_MOST_CITED_REDUCE extends Reducer<Text, IntWritable, Text, IntWritable> {

        int max = 0;
        Text maxWord = new Text();
        /*
         * REDUCE RETURN UNIQUE TITLE-ONE PER BLOCK. SO WE SHOULD GET UNIQUE IN REDUCE
         */

        public void reduce(Text key, Iterable<IntWritable> values, Context context)
                throws IOException, InterruptedException {

            int sum = 0;

            for (IntWritable val : values) {

                sum += val.get();

            } // SUM PER TAG
            if (sum > max) {
                max = sum;
                maxWord.set(key);
            }
```

MAP

REDUCE

choirak0805@kmubigdata-cluster-m: ~/HW_03_MR/Q06 - Chrome

https://ssh.cloud.google.com/projects/sound-decoder-221702/zones/asia-northeast1-c/instances/kmubigdata-cluster-m?authuser=1&hl=ko&projectNumber=176045424343

GNU nano 2.7.4 File: COUNT_MOST_CITED.java

```
                maxWord.set(key);
            }

            // context.write(key,new IntWritable (sum));
        }

        protected void cleanup(Context context )throws IOException, InterruptedException {
            context.write(maxWord, new IntWritable(max));
        }
    }
}
```

REDUCE

[Read 98 lines]

^G Get Help ^O Write Out ^W Where Is ^K Cut Text ^J Justify ^C Cur Pos ^Y Prev Page M-\ First Line M-W WhereIs Next ^^ Mark Text M-} Indent Text
^X Exit ^R Read File ^\ Replace ^U Uncut Text ^T To Spell ^_ Go To Line ^V Next Page M-/ Last Line M-] To Bracket M-^ Copy Text M-{ Unindent Text

오전 1:42
2018-11-27

```
        HDFS: Number of write operations=3
    Job Counters
        Launched map tasks=1
        Launched reduce tasks=1
        Data-local map tasks=1
        Total time spent by all maps in occupied slots (ms)=23964
        Total time spent by all reduces in occupied slots (ms)=17920
        Total time spent by all map tasks (ms)=5991
        Total time spent by all reduce tasks (ms)=4480
        Total vcore-milliseconds taken by all map tasks=5991
        Total vcore-milliseconds taken by all reduce tasks=4480
        Total megabyte-milliseconds taken by all map tasks=12269568
        Total megabyte-milliseconds taken by all reduce tasks=9175040
    Map-Reduce Framework
        Map input records=465565
        Map output records=465565
        Map output bytes=7480657
        Map output materialized bytes=8411831
        Input split bytes=110
        Combine input records=0
        Combine output records=0
        Reduce input groups=38643
        Reduce shuffle bytes=8411831
        Reduce input records=465565
        Reduce output records=1
        Spilled Records=931130
        Shuffled Maps =1
        Failed Shuffles=0
        Merged Map outputs=1
        GC time elapsed (ms)=214
        CPU time spent (ms)=5920
        Physical memory (bytes) snapshot=804356096
        Virtual memory (bytes) snapshot=6967902208
        Total committed heap usage (bytes)=717750272
    Shuffle Errors
        BAD_ID=0
        CONNECTION=0
        IO_ERROR=0
        WRONG_LENGTH=0
        WRONG_MAP=0
        WRONG_REDUCE=0
    File Input Format Counters
        Bytes Read=16603996
    File Output Format Counters
        Bytes Written=12
choirak0805@kmubigdata-cluster-m:~/HW_03_MR/Q06$ hdfs dfs -cat /hw03/Q06/final_output/*
18/11/24 16:05:27 INFO gcs.GoogleHadoopFileSystemBase: GHFS version: 1.6.10-hadoop2
sci-fi  3384
<-RESULT
choirak0805@kmubigdata-cluster-m:~/HW_03_MR/Q06$
```