맵 : 데이터를 쪼개서 나눠주는 것
리듀스 : 쪼개진 데이터를 합치는 것

ex)
read a book
write a book

=> 맵으로 변환(key:line number, value:문장)
1, read a book
2, write a book

=>정렬과 병합(key : 단어, value : 단어수)
<read, 1>
<a, 1>
<book, 1>

<write, 1>
<a, 1>
<book, 1>

=>Reduce(key:단어, value:단어수의 리스트)
<read, (1)>
<a, (1,1)>
<book, (1,1)>
<write, (1)>

=>실행결과 (key:단어, value:리스트의 합계)
<read, 1>
<a, 2>
<book, 2>
<write, 1>

---

## 나. 맵리듀스 프로그래밍 요소

### 1) 데이터 타입

맵리듀스 프로그램에서 키와 값으로 사용되는 모든 데이터 타입은 반드시 WritableComparable 인터페이스를 구현해야 함

1) WritableComparable 인터페이스를 구현한 Wrapper 클래스 목록

| 클래스명 | 데이터 타입 |
|---|---|
| BooleanWritable | Boolean |
| ByteWritable | 단일 Byte |
| DoubleWritable | Double |
| FloatWritable | Float |
| IntWritable | Int |
| LongWritable | Long |
| TextWritable | UTF-8 형식의 문자열 |
| NullWritable | 데이터 값이 필요없을 경우 사용 |

2) InputFormat

| InputFormat | 기능 |
|---|---|
| TextInputFormat | 텍스트 파일을 분석할 때 사용<br>개행문자를 기준으로 레코드 분류<br>Key : Line Number(LongWritable 타입)<br>Value : Line의 내용(Text 타입) |
| KeyValueTextInputFormat | 텍스트 파일을 분석할 때<br>라인 번호가 아닌 임의의 key를 사용 |
| NLineInputFormat | 텍스트 파일의 라인수를 제한할 때 사용 |
| DelegatingInputFormat | 여러 개의 서로 다른 입력 포맷을 사용할 경우<br>각 경로에 대한 작업을 위임 |
| CombineFileInputFormat | 여러 개의 파일을 입력받을 경우 사용 |
| SequenceFileInputFormat | SequenceFile을 입력받을 경우 사용<br>SequenceFile - 바이너리 형태의 키와 값의<br>목록으로 구성된 텍스트 파일 |
| SequenceFileAsBinaryInputFormat | SequenceFile의 키와 값을 임의의 바이너리<br>객체로 변환하여 사용 |
| SequenceFileAsTextInputFormat | SequenceFile의 키와 값을 Text 객체로 변환해서<br>사용 |

2) Mapper

key와 value로 구성된 입력 데이터를 전달받아 데이터를 가공하고 분류해 새로운 데이터 목록을 생성

3) Partitioner

맵 태스트의 출력 데이터가 어떤 리듀스 태스트로 전달될지 결정

4) Reducer

Map Task의 출력 데이터를 입력 데이터로 전달받아 집계 연산 수행

5) Combiner

Mapper의 출력 데이터를 입력 데이터로 전달받아 연산을 수행하여 Shuffle할 데이터의 크기를 줄일 경우 사용

Shuffle : Map Task와 Reduce Task 사이의 데이터 전달 과정

6) OutputFormat

| OutputFormat | 기능 |
|---|---|
| TextOutputFormat | 텍스트 파일에 레코드를 출력할 때 사용Key와 Value의 구분자는 탭 문자 |
| SequenceFileOutputFormat | SequenceFile을 출력물로 사용할 경우 |
| SequenceFileAsBinaryOutputFormat | 바이너리 포맷의 Key와 Value를 사용 |
| FilterOutputFormat | OutputFormat 클래스를 편리하게 사용할 수 있는 method 제공 |
| NullOutputFormat | 출력 데이터가 없을 때 사용 |

## 다. 실습 예제(WordCount)

### 1) count.MyMapper.java

```
package count;

import java.io.IOException;
import java.util.StringTokenizer;

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;

public class MyMapper 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 {
        StringTokenizer st = new StringTokenizer(value.toString());
        while (st.hasMoreTokens()) {
            word.set(st.nextToken());
            context.write(word, one);
        }
    }
}
```

```
package count;

import java.util.StringTokenizer;
import java.io.IOException;

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;

//Mapper<입력key자료형, 입력value자료형, 출력key자료형, 출력value자료형>
//line number, text가 input으로 들어온다.
//단어, 단어수가 output으로 나온다.
```

```
public class MyMapper extends Mapper <LongWritable, Text, Text, IntWritable>{
        //new IntWriable(1)임을 주의해야한다. 지정된 자료형을 사용해야한다.
         private final static IntWritable one = new IntWritable(1);
         private Text word = new Text();

         //라인번호, 텍스트 형식의 내용을 읽어서=>단어, 카운트 형식으로 변환한다.
         public void map(LongWritable key, Text value, Context context) throws
IOException, InterruptedException{
                  //value를 string형태로 변환하여 StringTokenizer에 넣는다.
                   StringTokenizer st = new StringTokenizer(value.toString());
                  //Token이 있는한, 하나씩 word에 set을 하고 이를 context 즉, 하둡에 지정된
문서 자료형에 적어준다.
                   while(st.hasMoreTokens()) {
                            word.set(st.nextToken());
                            context.write(word,  one);
                   }
         }
}
```

2) count.MyReducer.java

```
package count;

import java.io.IOException;

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;

public class MyReducer extends Reducer<Text, IntWritable, Text, IntWritable> {
    private IntWritable result = new IntWritable();
```

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

```
package count;


import java.io.IOException;

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;

//Reducer<입력 key자료형, 입력 value자료형, 출력 key자료형, 출력 value 자료형
public class MyReducer extends Reducer<Text, IntWritable, Text, IntWritable> {
        private IntWritable result = new IntWritable();
        //key, 리스트로 변환하는 코드
        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);
                context.write(key,  result);
        }
}
```

3) count.WordCount.java

```
package count;

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.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;
```

```
public class WordCount {
    public static void main(String[] args) throws Exception {
        Configuration conf = new Configuration();
        if (args.length != 2) {
            System.err.println("Usage: WordCount <input> <output>");
            System.exit(1);
        }
        Job job = Job.getInstance(conf, "WordCount");

        job.setJarByClass(WordCount.class);
        job.setMapperClass(MyMapper.class);
        job.setReducerClass(MyReducer.class);
        job.setInputFormatClass(TextInputFormat.class);
        job.setOutputFormatClass(TextOutputFormat.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);
        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));

        job.waitForCompletion(true);
    }
}
```

```
package count;

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.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;


public class WordCount {
        public static void main(String[] args) throws Exception{
                //Configuration은 하둡 환경설정 클래스이다.
                 Configuration conf = new Configuration();
                //입력 매개변수가 2개가 아니면, 에러메세지 출력및 프로그램 종료
                //입력 text파일이 1번째 매개변수이고, 출력파일 디렉토리 이름이 두번째
```

매개변수이다.

```
                if(args.length != 2) {
                        //에러메세지 출력
                        System.err.println("Usage : WrodCount <input> <output>");
                        //프로그램 종료
                        System.exit(1);
                }

                //HDFS(하둡 분산 파일 시스템)에 새로운 작업 할당
                //Job이라는 것은 MapReduce job이다.
                //하둡 분산 파일 시스템이 Job을 하나 만들어 준다는 의미이고,
WordCount는 job이름이다.
                Job job = Job.getInstance(conf, "WordCount");

                //실행할 클래스 이름지정
                job.setJarByClass(WordCount.class);
                //mapper 클래스 이름 지정
                job.setMapperClass(MyMapper.class);
                //reducer 클래스 이름 지정
                job.setReducerClass(MyReducer.class);

                //입력 자료형 지정
                job.setInputFormatClass(TextInputFormat.class);
                //출력 자료형 지정
                job.setOutputFormatClass(TextOutputFormat.class);
                //(둘다 Text로 지정하였다.)

                //key의 자료형 지정(Text)
                job.setOutputKeyClass(Text.class);
                //value의 자료형 지정(숫자)
                //int or Integer는 사용하면 안된다. 하둡에서 지정된 IntWritable로 사용한다.
                job.setOutputValueClass(IntWritable.class);

                //입력 파일의 경로
                FileInputFormat.addInputPath(job, new Path(args[0]));
                //출력 파일의 경로
                FileOutputFormat.setOutputPath(job, new Path(args[1]));
                //이 두개는 매개변수로 들어올 것이다.

                //분석 작업이 끝날 때 까지 대기
                job.waitForCompletion(true);
        }
}
```

4) Hadoop.jar 파일로 export

리눅스 master node의 /home/centos/source 디렉토리에 복사

5) 텍스트파일 생성

/home/centos/input.txt 파일 생성

```
gedit  /home/centos/input.txt
```

```
read a book
write a book
```

6) 텍스트파일을 하둡시스템에 업로드

```
hdfs dfs -rm input.txt
hdfs dfs -put /home/centos/input.txt  input.txt
hdfs dfs -ls
```

7) WordCount 실행

```
hadoop jar /home/centos/source/Hadoop.jar count.WordCount input.txt wordcount_output
```

8) 결과 확인

```
hdfs dfs -cat wordcount_output/part-r-00000
```

```
[root@master ~]# hdfs dfs -cat wordcount_output/part-r-00000
a       2
book    2
read    1
write   1
```

(기존폴더 삭제)

```
hdfs dfs -rm -r wordcount_output
```

9) 웹브라우저에서 확인

http://master:50070

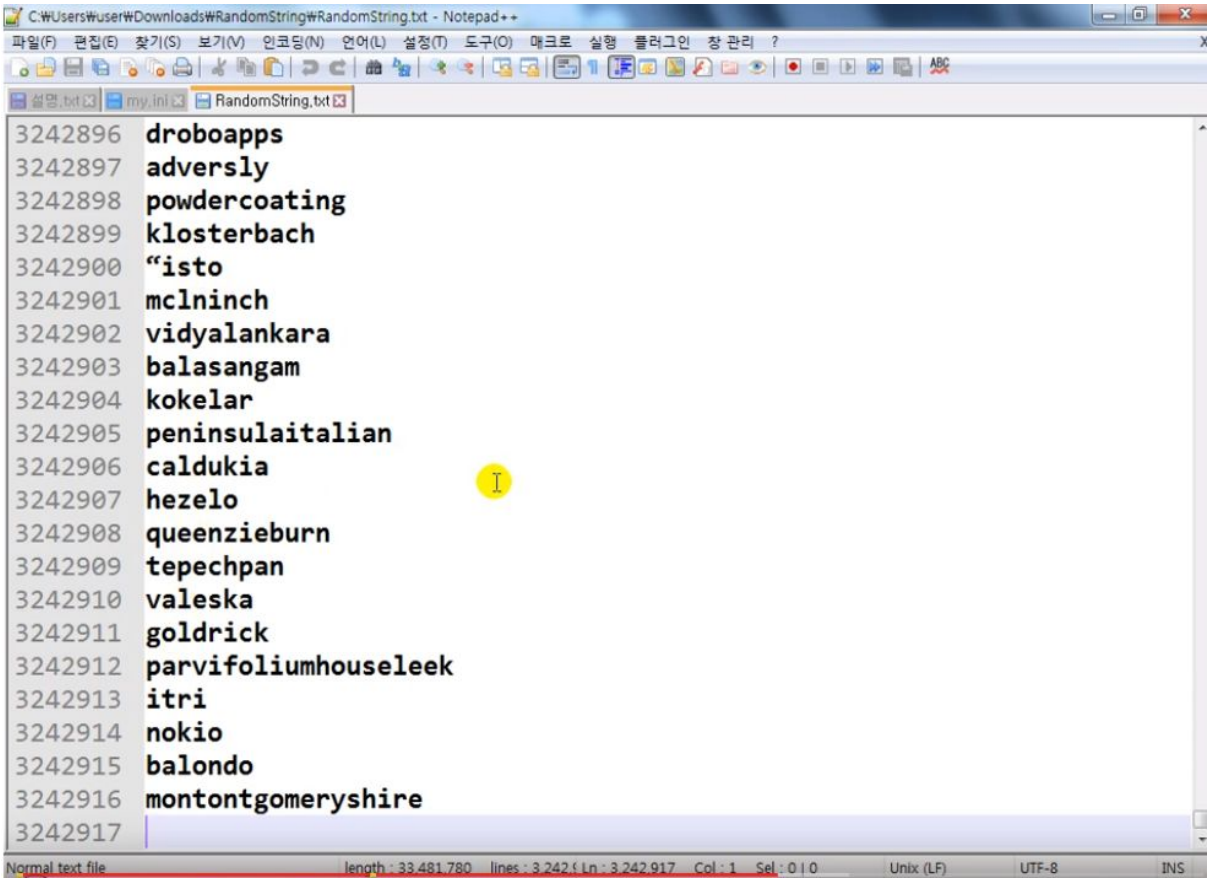
---

http://mannaedu.com/bbs/board.php?bo_table=pds&wr_id=71&sca=%EB%8D%B0%EC%9D%B4%ED%84%B0%ED%8C%8C%EC%9D%BC

에서 RandomString.txt파일 을 받는다.

압축 풀기.

약간 큰 단어들이 적힌




약 3백만건 정도의 단어들이 있다.

이러한 데이터를 하둡으로 읽어서 소팅을 해보자.

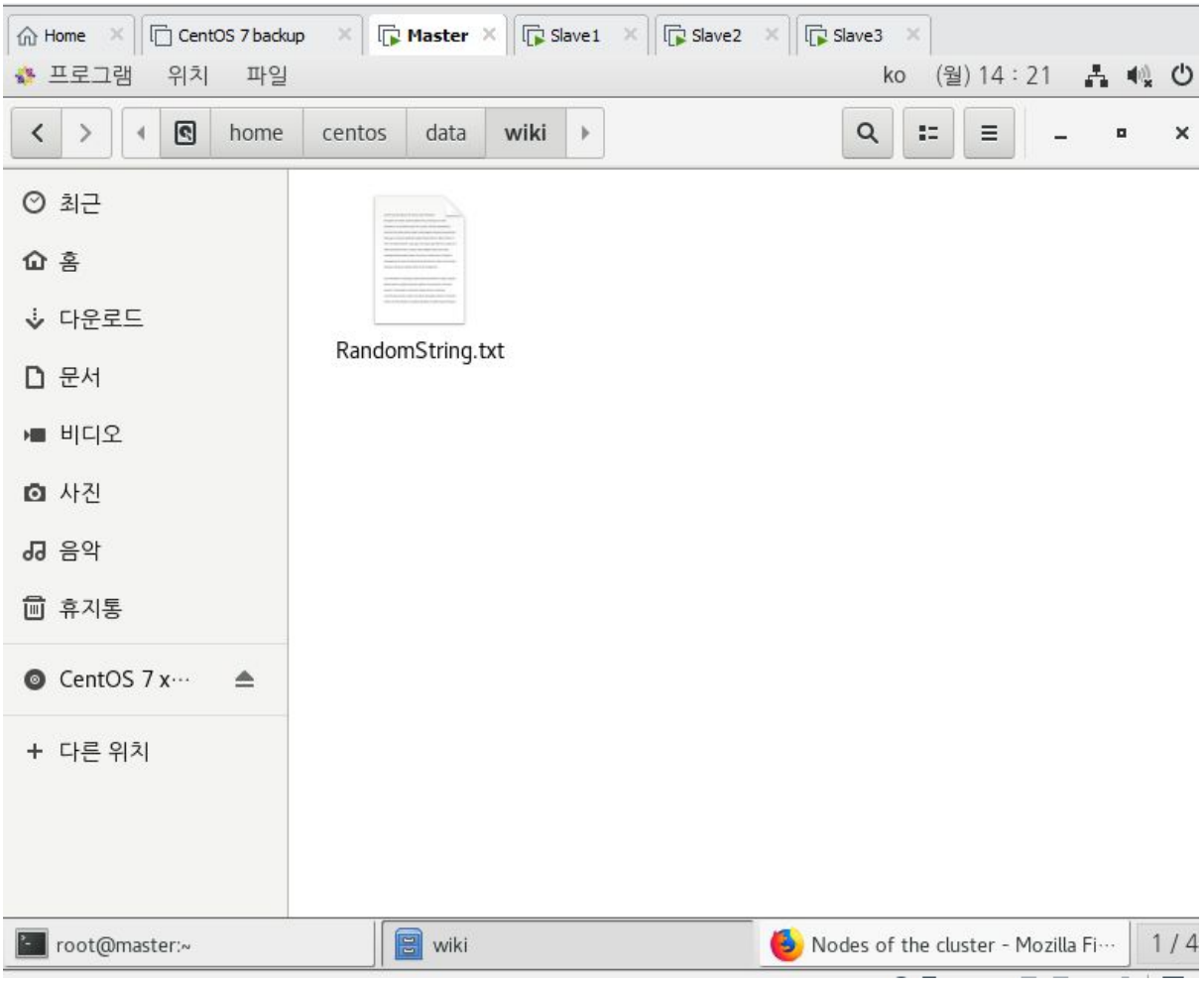
DB로 읽어들여서 order by를 한다면, 이는 굉장히 오래걸린다.
하지만 수십대의 hadoop분산처리를 하면 굉장히 빠르게 처리가 가능하다.

## 라. 실습 예제(StringSort)

3242916 라인의 랜덤 문자열을 오름차순으로 정렬

1) RandomString.txt 파일을 리눅스 서버에 복사

복사 위치 : /home/centos/data/wiki/RandomString.txt



2) sort.StringSort.java

```
package sort;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.SequenceFile;
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.KeyValueTextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.SequenceFileOutputFormat;

public class StringSort {
    public static void main(String[] args) throws Exception {
        Configuration conf = new Configuration();
        Job job = Job.getInstance(conf, "StringSort");
        job.setJarByClass(StringSort.class);
        //mapper 지정(기본 mapper를 사용, 입력되는 레코드가 그대로 출력
레코드가 됨)
        job.setMapperClass(Mapper.class);
```

…

```
package sort;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.SequenceFile;
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.KeyValueTextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.SequenceFileOutputFormat;

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

                //실행클래스
                Job job = Job.getInstance(conf, "StringSort");

                //mapper 지정(기본 mapper를 사용, 입력되는 레코드가 그대로 출력
레코드가 됨)
                job.setMapperClass(Mapper.class);

                //reducer 지정(기본 reducer를 사용)
                //맵에서 출력되는 것이 그대로 리듀스의 출력이 됨,
                //리듀스 단에서 내부적으로 sort가 이루어짐
                job.setReducerClass(Reducer.class);
```

```
                //맵 출력과 리듀스 출력의 key, value 타입 설정
                job.setMapOutputKeyClass(Text.class);
                job.setMapOutputValueClass(Text.class);
                job.setOutputKeyClass(Text.class);
                job.setOutputValueClass(Text.class);

                //reduce의 수를 1로 설정
                // (모든 맵의 출력이 하나의 리듀스 테스트로 가게됨,
                //  레코드 갯수가 그대로 이며 sort만 하게 됨)
                job.setNumReduceTasks(1);

                job.setInputFormatClass(KeyValueTextInputFormat.class);
                job.setOutputFormatClass(SequenceFileOutputFormat.class);

                //입력 파일
                FileInputFormat.addInputPath(job,  new Path(args[0]));
                //출력 디렉토리(사이즈를 줄이기 위해 시퀀스 포맷 사용,
                //60% 정도 사이즈 감소)
                //일종의 압축파일이라고 생각하면 된다.
                SequenceFileOutputFormat.setOutputPath(job,  new Path(args[1]));

                //블록 단위 압축(레코드 단위로 압축)
                SequenceFileOutputFormat.setOutputCompressionType(job,
SequenceFile.CompressionType.BLOCK);


                job.waitForCompletion(true);
        }
}
```

3) Hadoop.jar로 export

/home/centos/source 디렉토리에 복사

4) 데이터 분석 실행

```
hdfs dfs -put /home/centos/data/wiki/RandomString.txt   /input/RandomString.txt

hadoop jar /home/centos/source/Hadoop.jar sort.StringSort
/input/RandomString.txt   /random-string-output

hdfs dfs -ls /random-string-output
```

cat 명령어가 아닌 text 명령어로 읽어야 함

```
hdfs dfs -text /random-string-output/part-r-00000
```