MapReduce Programming

MapReduce Programming

- Compiling & Running the Hadoop wordcount example

```
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;
```

MapReduce Programming

- Compiling & Running the Hadoop wordcount example

```
public class WordCount {
  public static class TokenizerMapper
       extends Mapper<Object, Text, Text, IntWritable>{
    private final static IntWritable one = new IntWritable(1);
    private Text word = new Text();
    public void map(Object key, Text value, Context context
                    ) throws IOException, InterruptedException {
      StringTokenizer itr = new StringTokenizer(value.toString());
      while (itr.hasMoreTokens()) {
        word.set(itr.nextToken());
        context.write(word, one);
      }
    }
  }
```

MapReduce Programming

- Compiling & Running the Hadoop wordcount example

```
public static class IntSumReducer
      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);
   }
 }
```

MapReduce Programming

- Compiling & Running the Hadoop wordcount example

```
public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();
    Job job = Job.getInstance(conf, "word count");
    job.setJarByClass(WordCount.class);
    job.setMapperClass(TokenizerMapper.class);
    job.setCombinerClass(IntSumReducer.class);
    job.setReducerClass(IntSumReducer.class);
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);
    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));
    System.exit(job.waitForCompletion(true) ? 0 : 1);
  }
}
```

MapReduce Programming

- (input) <k1, v1> -> map -> <k2, v2> -> combine -> <k2, v2> -> reduce -> <k3, v3>
- (output)

MapReduce Programming

```
public void map(Object key, Text value, Context context
                ) throws IOException, InterruptedException {
    StringTokenizer itr = new StringTokenizer(value.toString());
    while (itr.hasMoreTokens()) {
      word.set(itr.nextToken());
      context.write(word, one);
    }
  }
```

< Hello, 1>

< World, 1>

< Bye, 1>

< World, 1>

< Hello, 1>

< Hadoop, 1>

< Goodbye, 1>

< Hadoop, 1>

MapReduce Programming

- job.setMapperClass(TokenizerMapper.class);
- job.setCombinerClass(IntSumReducer.class);
- job.setReducerClass(IntSumReducer.class);

- < Bye, 1>
- < Hello, 1>
- < World, 2>
- < Goodbye, 1>
- < Hadoop, 2>
- < Hello, 1>

- $ javac -cp `hadoop classpath` -d wordcount_classes WordCount.java

HDFS API

- hdfs dfs로 작성할 수 있는 FileSystem Shell 명령을 실행하는 API

- Configuration 클래스
  - addResurce, clear, get, getResource, set
- Path 클래스
  - CUR_DIR, SEPARATOR, SEPARATOR_CHAR, WINDOWS, PATH, getFileSystem, isRoot
- FileSystem 클래스
  - Append, close, copyFromLocalFile, copyFromLocalFile, copyToLocalFile, create, delete, exists, get, getFileStatus, isDirectory, newInstance, open
- FileStatus 클래스
  - getBlockSize, getGroup, getLen, getOwner, get isDirectory, isFile

MapReduce 기본 데이터 타입

- ArrayWritable : 배열객체
- BooleanWritable : Boolean
- ByteWritable : Byte
- DoubleWritable : Double
- FloatWritable : Float
- IntWritable : Integer
- LongWritable : Long
- MapWritable : Map 객체
- NullWritable : Null 객체, Key나 Value값에 지정할 객체가 없는 경우
- Text : 문자열

사용자 타입정의

- public class MyTypeWritable implements WritableComparable<MyTypeWritable>{

```
privagte String name;
private int age;
public void write(DataOutput out) throws IOException {
        Out.writeString(name);
        Out.writeInt(age);
}
public void readFields(DataInput in) throw IOException {
        Name = in.readString()
        Age = in readInt()
}
public int compareTo(MyTypeWrtable key) {
        Int result = name.compareTo(key.name)
        Int(result == 0){
                Int thisAge = this.age;
                Int thatAge =key.agg:
                ....
        }
}
```

Map 클래스

- MapReduce태스크의 Map작업은 org.apache.Hadoop.Map/Reduce<KEYIN, VALUEIN, KEYOUT, VALUEOUT>객체를 상속하고 Mapper클래스의 API함수를 overriding형태로 구현함
  - protected void cleanup(org.apache.Hadoop.Map/Reduce.Mapper.Context context)
    - Map 태스크 작업이 종료될때 한번 호출됨
  - protected void map(org.apache.Hadoop.Map/Reduce.Mapper.Context context)
    - 각각의 Key/Value작업을 처리할 때 한번씩 호출됨
  - protected void setup(org.apache.Hadoop.Map/Reduce.Mapper.Context context)
    - Map태스크 작업이 시작될때 한번 호출됨

Map 클래스

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

        private final static IntWritable outputValue = new IntWritable(1);
        private Text outputKey = new Text();
@Override
 public void map(LongWritable key, Text value, Context context) throws IOException,
InterruptedException {
        String line = value.toString();
        StringTokenizer tokenizer = new StringTokenizer(line);

        while (tokenizer.hasMoreTokens()) {
                outputKey.set(tokenizer.nextToken());
                context.write(outputKey, outputValue );
                        }
                }
        }
}
```

Reduce 클래스

- MapReduce 태스크의 Reduce작업은 org.apache.Hadoop.Map/Reduce.Reducer<KEYIN, VALUEIN, KEYOUT, VALUEOUT>객체를 상속하고 Reducer클래스의 API함수를 overriding형태로 구현함
- Reduce클래스는 Mapper에서 만들어진 Key-Value출력값을 Shuffle하고 Sorting한 다음 Reduce작업을 실행함
  - protected void cleanup(org.apache.Hadoop.Map/Reduce.Mapper.Context context)
  - Reduce 태스크 작업이 종료될때 한번 호출됨
  - protected void map(org.apache.Hadoop.Map/Reduce.Mapper.Context context)
  - 각각의 Key/Value작업을 처리할 때 한번씩 호출됨
  - protected void setup(org.apache.Hadoop.Map/Reduce.Mapper.Context context)
  - Reduce태스크 작업이 시작될때 한번 호출됨

Reduce 클래스

```
public static class MyReducer extends Reducer<Text, IntWritable, Text,
IntWritable> {


@Override
public void reduce(Text key, Iterable<IntWritable> values, Context context)
        throws IOException, InterruptedException {
                int sum = 0;
                for (IntWritable value : values) {
            sum += value.get();
                }
                context.write(key, new IntWritable(sum));
            }
        }
}
```

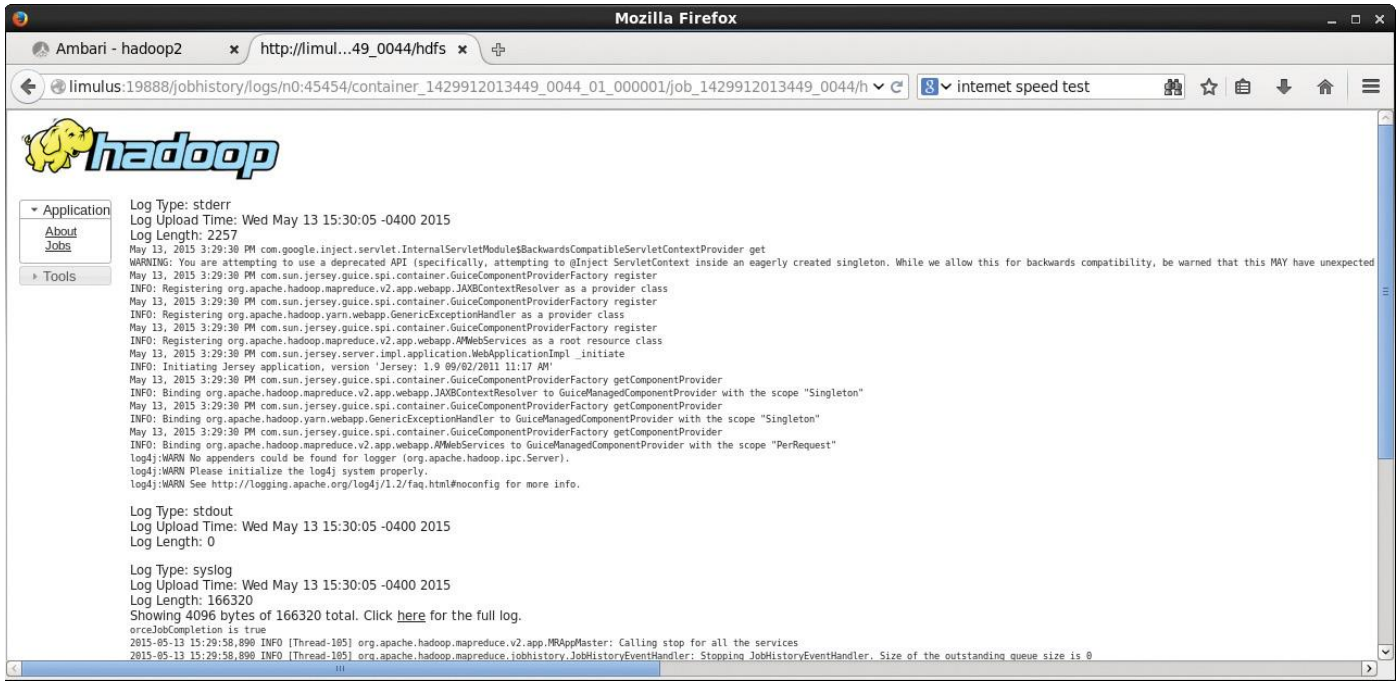
Debugging MapReduce

- Listing, Killing, Job Status
  - 맵핑 된 작업 명령을 사용
  - 가장 많이 사용되는 옵션은 -list, -kill 및 -status
  - Yarn application 명령을 사용하여 클러스터에서 실행중인 모든 응용 프로그램을 제어 할 수 있음

- Hadoop Log Management
  - MapReduce 로그는 매퍼와 축소에 대한 정보를 제공함
  - 로그 출력은 응용 프로그램에 대해 stdout, stderr 및 syslog (Hadoop 시스템 메시지)의 세 파일로 구성됨
  - 로그 저장에는 두 가지 모드가 있음
    - 첫 번째 (그리고 가장 좋은) 방법은 로그 집계를 사용하는 것임
    - 이 모드에서 로그는 HDFS로 집계되며 YARN ResourceManager 사용자 인터페이스에 표시되거나 yarn logs 명령으로 검사 할 수 있음

Debugging MapReduce

- 로그 집계가 사용 가능하지 않으면 로그는 맵퍼 또는 리듀서가 실행 된 클러스터 노드에 로컬로 배치
- 집계되지 않은 로컬 로그의 위치는 yarn-site.xml 파일의 yarn.nodemanager.log-dirs 등록 정보에서 제공함
- 로그 집계가 없으면 작업에서 사용하는 클러스터 노드를 기록한 다음 로그 파일을 노드에서 직접 가져와야함
- 로그 집계를 적극 권장



Debugging MapReduce

- YARN 로그 집계 사용
  - Apache Hadoop이 공식 Apache Hadoop 소스에서 설치된 경우 다음 설정을 사용하면 시스템에서 로그 집계가 설정됨
  - Ambari 또는 다른 관리 도구를 사용하는 경우 해당 도구를 사용하여 설정을 변경가능함
  - Apache Ambari를 사용하는 경우 YARN 서비스 구성 탭에서 yarn.log-aggregation-enabl을 확인 필요 기본 설정은 사용 가능함

Debugging MapReduce

- $ hdfs dfs -mkdir -p /yarn/logs
- $ hdfs dfs -chown -R yarn:hadoop /yarn/logs
- $ hdfs dfs -chmod -R g+rw /yarn/logs

```
<property>
  <name>yarn.nodemanager.remote-app-log-dir</name>
  <value>/yarn/logs</value>
</property>
<property>
  <name>yarn.log-aggregation-enable</name>
  <value>true</value>
</property>
```