Lab Module 3: Compiling and Running the Word Count Program with and without Skipping Patterns

Objective

Learn how to compile and run a custom Word Count MapReduce program in Hadoop. The lab will demonstrate how to run the program both with and without skipping specific patterns (e.g., stopwords) during the word count process.

Topics Covered

- Compiling a Word Count MapReduce Program
- Running Word Count without Skipping Patterns
- Running Word Count with Skipping Patterns
- Analyzing and Comparing the Output

Lab Activities

Activity 1: Set Up the Environment

1. **Set Up Hadoop:**
    - Ensure that Hadoop is installed and running on your machine.
    - Verify the Hadoop services by running:
      jps
    - Services such as NameNode, DataNode, ResourceManager, and NodeManager should be running.
2. **Prepare the Input Data:**
    - Create a sample text file named input.txt containing text data:

      Hadoop is an open-source framework. Hadoop is used for processing large datasets. Hadoop MapReduce is the processing component of Hadoop.

    - Upload the input.txt file to HDFS:
      hdfs dfs -mkdir /user/your_username/wordcount_input
      hdfs dfs -put input.txt /user/your_username/wordcount_input

Activity 2: Compile the Word Count Program

1. **Write the Word Count Java Program:**
   Create a Java program WordCount.java for counting words:

```
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;

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);
    }
  }
 }

 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);
  }
 }

 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);
 }
```

```
}
```

2. **Compile the Program:**
   - Compile the Java program:
     `javac -classpath `hadoop classpath` -d . WordCount.java`
   - Create the jar file:
     `jar cf wordcount.jar WordCount*.class`

Activity 3: Run Word Count Without Skipping Patterns

1. **Run the Word Count Program:**
   Execute the Word Count job without skipping any patterns:

```
hadoop jar wordcount.jar WordCount /user/your_username/wordcount_input /user/your_username/wordcount_output_no_skip
```

2. **View the Output:**
   - Check the output directory:
     `hdfs dfs -cat /user/your_username/wordcount_output_no_skip/part-r-00000`
   - The output will display the count of each word in the input file.

Activity 4: Modify Program to Skip Specific Patterns (e.g., Stopwords)

1. **Modify the Mapper to Skip Patterns:**
   - Create the WordCount2.java program to skip common stopwords (like "is", "an", "for"):

```
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.net.URI;
import java.util.ArrayList;
import java.util.HashSet;
import java.util.List;
import java.util.Set;
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.mapreduce.Counter;
import org.apache.hadoop.util.GenericOptionsParser;
import org.apache.hadoop.util.StringUtils;

public class WordCount2 {

 public static class TokenizerMapper
      extends Mapper<Object, Text, Text, IntWritable>{
```

```
  static enum CountersEnum { INPUT_WORDS }

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

  private boolean caseSensitive;
  private Set<String> patternsToSkip = new HashSet<String>();

  private Configuration conf;
  private BufferedReader fis;

  @Override
  public void setup(Context context) throws IOException,
      InterruptedException {
    conf = context.getConfiguration();
    caseSensitive = conf.getBoolean("wordcount.case.sensitive", true);
    if (conf.getBoolean("wordcount.skip.patterns", false)) {
      URI[] patternsURIs = Job.getInstance(conf).getCacheFiles();
      for (URI patternsURI : patternsURIs) {
        Path patternsPath = new Path(patternsURI.getPath());
        String patternsFileName = patternsPath.getName().toString();
        parseSkipFile(patternsFileName);
      }
    }
  }

  private void parseSkipFile(String fileName) {
    try {
      fis = new BufferedReader(new FileReader(fileName));
      String pattern = null;
      while ((pattern = fis.readLine()) != null) {
        patternsToSkip.add(pattern);
      }
    } catch (IOException ioe) {
      System.err.println("Caught exception while parsing the cached file '"
          + StringUtils.stringifyException(ioe));
    }
  }

  @Override
  public void map(Object key, Text value, Context context
                  ) throws IOException, InterruptedException {
    String line = (caseSensitive) ?
        value.toString() : value.toString().toLowerCase();
    for (String pattern : patternsToSkip) {
      line = line.replaceAll(pattern, "");
    }
    StringTokenizer itr = new StringTokenizer(line);
    while (itr.hasMoreTokens()) {
      word.set(itr.nextToken());
      context.write(word, one);
      Counter counter = context.getCounter(CountersEnum.class.getName(),
          CountersEnum.INPUT_WORDS.toString());
      counter.increment(1);
    }
  }
}

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);
  }
}

public static void main(String[] args) throws Exception {
  Configuration conf = new Configuration();
  GenericOptionsParser optionParser = new GenericOptionsParser(conf, args);
  String[] remainingArgs = optionParser.getRemainingArgs();
  if ((remainingArgs.length != 2) && (remainingArgs.length != 4)) {
    System.err.println("Usage: wordcount <in> <out> [-skip skipPatternFile]");
    System.exit(2);
  }
  Job job = Job.getInstance(conf, "word count");
  job.setJarByClass(WordCount2.class);
  job.setMapperClass(TokenizerMapper.class);
  job.setCombinerClass(IntSumReducer.class);
  job.setReducerClass(IntSumReducer.class);
  job.setOutputKeyClass(Text.class);
  job.setOutputValueClass(IntWritable.class);

  List<String> otherArgs = new ArrayList<String>();
  for (int i=0; i < remainingArgs.length; ++i) {
    if ("-skip".equals(remainingArgs[i])) {
      job.addCacheFile(new Path(remainingArgs[++i]).toUri());
      job.getConfiguration().setBoolean("wordcount.skip.patterns", true);
    } else {
      otherArgs.add(remainingArgs[i]);
    }
  }
  FileInputFormat.addInputPath(job, new Path(otherArgs.get(0)));
  FileOutputFormat.setOutputPath(job, new Path(otherArgs.get(1)));

  System.exit(job.waitForCompletion(true) ? 0 : 1);
 }
}
```

Compile the Program:

- Compile the modified Java program:
  `javac -classpath `hadoop classpath` -d . WordCount2.java`
- Create jar file:
  `jar cf wordcount_skip.jar WordCount*.class`

Activity 5: Run Word Count with Skipping Patterns

1. **Run the Modified Word Count Program:**

- Execute the Word Count job with the skip logic:
  `hadoop jar wc.jar WordCount2 -Dwordcount.case.sensitive=true /user/joe/wordcount/input /user/joe/wordcount/output -skip /user/joe/wordcount/patterns.txt`

2. **View the Output:**
   - Check the output directory:
     `hdfs dfs -cat /user/your_username/wordcount_output_skip/part-r-00000`
   - The output will display the count of each word, excluding the skipped patterns (stopwords).

Activity 6: Compare the Outputs

1. **Analyze the Results:**
   - Compare the outputs from both runs (with and without skipping patterns) to understand the impact of pattern skipping on word counts.
   - Discuss why certain words are excluded and how it affects the final word count.

Assessment

1. **Practical Exercise:**
   - Students are required to modify the Word Count program to skip a different set of patterns (e.g., custom stopwords or specific phrases) and analyze the output.

Conclusion

This lab module provides hands-on experience in compiling and running a Hadoop MapReduce program, first without any modification and then with custom logic to skip specific patterns. This exercise will help you understand how to customize MapReduce jobs based on specific requirements and the implications of these customizations on data processing results.