Data Intensive Computing – Handout 8

Hadoop

Hadoop 1.2.1 is installed in `/HADOOP` directory. The JobTracker web interface is available at `http://dlrc:50030`, the NameNode web interface is available at `http://dlrc:50070`.

To conveniently access the web interfaces remotely, you can use `ssh -D localhost:2020 -p 11422 ufallab.ms.mff.cuni.cz` to open SOCKS5 proxy server forwarding requests to the remove site, and use it as a proxy server in a browser, for example as `chromium --proxy-server=socks://localhost:2020`.

Java Interface

Running Java Jobs

Java jobs are executing using `hadoop jar filename.jar` command. The additional arguments are passed to the `filename.jar` program. Usually the arguments are `[options] input_path output_pat` with the following options:

- `-D mapred.map.tasks=N` set specified number of map tasks
- `-D mapred.reduce.tasks=N` set specified number of reduce tasks, can be 0
- `-D mapreduce.inputformat.class=cls` set specified input format, can be one of
  - `org.apache.hadoop.mapreduce.lib.input.TextInputFormat`
  - `org.apache.hadoop.mapreduce.lib.input.SequenceFileInputFormat`
  - `org.apache.hadoop.mapreduce.lib.input.KeyValueTextInputFormat`
- `-D mapreduce.outputformat.class=cls` set specified input format, can be one of
  - `org.apache.hadoop.mapreduce.lib.output.TextOutputFormat`
  - `org.apache.hadoop.mapreduce.lib.output.SequenceFileOutputFormat`

Running Java Jobs Locally

Java jobs can be executed locally by using `hadoop-local` command. In that case, mappers and reducers are executed sequentially, directly in the running process.

Manupulating HDFS

The HFDS content can be manupulated using `hadoop fs` command, which prints nice usage when executed without arguments. Some simple commands:

- `hadoop fs -ls /`
- `hadoop fs -rmr /users/straka/test /`
- `hadoop fs -get /users/straka/test . /`

Java Examples

All mentioned examples are available in `/home/straka/java/examples`.

Grep AB

GrepAB.java

```
import java.io.IOException;

import org.apache.hadoop.conf.*;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapreduce.*;
import org.apache.hadoop.mapreduce.lib.input.*;
import org.apache.hadoop.mapreduce.lib.output.*;
import org.apache.hadoop.util.*;

public class GrepAB extends Configured implements Tool {
  // Mapper
  public static class TheMapper extends Mapper<Text, Text, Text, Text>{
    public void map(Text key, Text value, Context context) throws IOException,
        InterruptedException {
      if (key.toString().startsWith("AB"))
        context.write(key, value);
    }
  }

  // Job configuration
  public int run(String[] args) throws Exception {
    if (args.length < 2) {
      System.err.printf("Usage: %s.jar in-path out-path\n",
          this.getClass().getName());
      return 1;
    }

    Job job = new Job(getConf(), this.getClass().getName());
    job.setJarByClass(this.getClass());
    job.setMapperClass(TheMapper.class);
    job.setMapOutputKeyClass(Text.class);
    job.setMapOutputValueClass(Text.class);

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

    return job.waitForCompletion(true) ? 0 : 1;
  }

  // Main method
  public static void main(String[] args) throws Exception {
    int res = ToolRunner.run(new GrepAB(), args);
    System.exit(res);
  }
}
```

GrepAB.sh

```
hadoop fs -rmr /users/$USER/GrepAB
hadoop jar GrepAB.jar \
  -D mapreduce.inputformat.class=org.apache.hadoop.mapreduce.lib.input.SequenceFileInp
  -D mapred.reduce.tasks=0 \
  /data/wiki/cs-medium /users/$USER/GrepAB
```

Compiling Examples

Examples can be compiled using either:

- supplied `Makefile`: just run `make` in the source directory and a `GrepAB.jar` is created. Use `make clean` to remove the compiled classes and java archive.
- ant: create a directory and copy `/home/straka/java/ant/build.xml` to it. Also create `src` subdirectory and copy `GrepAB.java` to it. Then run `ant` in the original directory to create `GrepAB.jar` in the `bin` directory.

  When using the `build.xml` on different class, update `target` property in `build.xml`.

  When compiling somewhere else than on `dlrc`, copy `/HADOOP/hadoop-core-1.2.1.jar` and update `hadoop-core.jar` property in `build.xml`.
- Eclipse: copy `build.xml` and `hadoop-core-1.2.1.jar` from `/home/straka/java/eclipse` to a local directory. Copy `GrepAB.java` to the `src` subdirectory. Then run Eclipse, select File – New – Project – Java Project from Existing Ant Buildfile, name the project, choose the `build.xml`, and check "Link to the buildfile in the file system". The new project will use the original directory as both source and output directory.

Hadoop Documentation

We use Hadoop 1.2.1. The API documentation can be found at `http://hadoop.apache.org/docs/r1.2.1/api/`.

Word Count example

Simple example using a reducer and combiner.

WordCount.java

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

import org.apache.hadoop.conf.*;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapreduce.*;
import org.apache.hadoop.mapreduce.lib.input.*;
import org.apache.hadoop.mapreduce.lib.output.*;
import org.apache.hadoop.util.*;

public class WordCount extends Configured implements Tool {
  // Mapper
  public static class TheMapper extends Mapper<Text, Text, Text, IntWritable>{
    private Text word = new Text();
    private IntWritable one = new IntWritable(1);

    public void map(Text key, Text value, Context context) throws IOException,
        InterruptedException {
      StringTokenizer st = new StringTokenizer(value.toString(), "␣
          \t\n\r\f,.:;?![]()'\"");
      while (st.hasMoreTokens()) {
        word.set(st.nextToken());
        context.write(word, one);
      }
    }
  }
```

```
  // Reducer
  public static class TheReducer extends Reducer<Text, IntWritable, Text,
      IntWritable> {
    private IntWritable sumWritable = new IntWritable();

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

  // Job configuration
  public int run(String[] args) throws Exception {
    if (args.length < 2) {
      System.err.printf("Usage: %s.jar in-path out-path\n",
          this.getClass().getName());
      return 1;
    }

    Job job = new Job(getConf(), this.getClass().getName());
    job.setJarByClass(this.getClass());
    job.setMapperClass(TheMapper.class);
    job.setMapOutputKeyClass(Text.class);
    job.setMapOutputValueClass(IntWritable.class);
    job.setCombinerClass(TheReducer.class);
    job.setReducerClass(TheReducer.class);
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);

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

    return job.waitForCompletion(true) ? 0 : 1;
  }

  // Main method
  public static void main(String[] args) throws Exception {
    int res = ToolRunner.run(new WordCount(), args);
    System.exit(res);
  }
}
```

WordCount.sh

```
hadoop fs -rmr /users/$USER/WordCount
hadoop jar WordCount.jar \
  -D mapreduce.inputformat.class=org.apache.hadoop.mapreduce.lib.input.SequenceFileInp
  /data/wiki/cs-medium /users/$USER/WordCount
```

Partitioner

Using Java Hadoop, partitioner can be specified easily:

Sort.java

```
import java.io.IOException;
```

```
import org.apache.hadoop.conf.*;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapreduce.*;
import org.apache.hadoop.mapreduce.lib.input.*;
import org.apache.hadoop.mapreduce.lib.output.*;
import org.apache.hadoop.util.*;

public class Sorting extends Configured implements Tool {
  public static class TheMapper extends Mapper<Text, Text, IntWritable, Text> {
    public void map(Text key, Text value, Context context) throws IOException,
        InterruptedException {
      // INSERT CODE HERE
    }
  }

  public static class ThePartitioner extends Partitioner<IntWritable, Text> {
    public int getPartition(IntWritable key, Text value, int numPartitions) {
      // INSERT CODE HERE
    }
  }

  public static class TheReducer extends Reducer<IntWritable, Text,
      IntWritable, Text> {
    public void reduce(IntWritable key, Iterable<Text> values, Context context)
        throws IOException, InterruptedException {
      // INSERT CODE HERE
    }
  }

  public int run(String[] args) throws Exception {
    if (args.length < 2) {
      System.err.printf("Usage: %s.jar in-path out-path",
          this.getClass().getName());
      return 1;
    }

    Job job = new Job(getConf(), this.getClass().getName());

    job.setJarByClass(this.getClass());
    job.setMapperClass(TheMapper.class);
    job.setMapOutputKeyClass(IntWritable.class);
    job.setMapOutputValueClass(Text.class);
    job.setPartitionerClass(ThePartitioner.class);
    job.setReducerClass(TheReducer.class);
    job.setOutputKeyClass(IntWritable.class);
    job.setOutputValueClass(Text.class);

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

    return job.waitForCompletion(true) ? 0 : 1;
  }

  public static void main(String[] args) throws Exception {
    int res = ToolRunner.run(new Sorting(), args);
    System.exit(res);
  }
}
```

Tasks

Solve the following tasks. Solution for each task is a Java Hadoop source processing the Wikipedia source data and producing required results.

Simple Tokenizer

You can use `StringTokenizer(sentence, " \t\n\r\f,.:;?![]()'\"")` as a simple tokenizer:

```
StringTokenizer st = new StringTokenizer(data, " \t\n\r\f,.:;?![]()'\"");
while (st.hasMoreTokens())
  System.out.println(st.nextToken());
```

Wikipedia Data

The Wikipedia Data available from `/dlrc_share/data/wiki/` are available also in HDFS:

- `/data/wiki/cs`: Czech Wikipedia data (Sep 2009), 195MB, 124k articles
- `/data/wiki/en`: English Wikipedia data (Sep 2009), 4.9GB, 2.9M articles

All data is stored in a record-compressed sequence files, with article names as keys and article texts as values, in UTF-8 encoding.

Task	*Points*	*Description*
`java_unique_words`	2	Create a list of unique words used in the articles using Java Hadoop. Convert them to lowercase to ignore case.
`java_anagrams`	2	Two words are anagrams if one is a letter permutation of the other (ignoring case). For a given input, find all anagram classes that contain at least A words. Output each anagram class on a separate line.
`java_sort`	3	You are given data consisting of (31-bit integer, string data) pairs. These are available in plain text format: • `/data/numbers-txt/numbers-small`: 3MB • `/data/numbers-txt/numbers-medium`: 184MB • `/data/numbers-txt/numbers-large`: 916MB You can assume that the integers are uniformly distributed. Your task is to sort these data, comparing the key numerically and not lexicographically. The lines in the output must be the same as in the input, only in different order. Your solution should work for TBs of data. For that reason, you must use multiple reducers. If your job is executed using r reducers, the output consists of r files, which when concatenated would produce sorted (key, value) pairs. In other words, each of the output files contains sorted (integer, data) pairs and all keys in one file are either smaller or larger than in other file. Your solution should work for any number of reducer specified. You can use the `Sort.java` example template.

Task	*Points*	*Description*
`java_nonuniform_sort`	4	Improve the `java_sort` to handle nonuniform data. You can use the following exponentially distributed data: • `/data/numbers-txt/nonuniform-small`: 3MB • `/data/numbers-txt/nonuniform-medium`: 160MB • `/data/numbers-txt/nonuniform-large`: 797MB Assume we want to produce r output files. One of the solutions is to perform two Hadoop jobs: • Go through the data and sample only a small fraction of the keys. As there are not so many of them, they can fit in one reducer. • Find best $r-1$ separators using the sampled data. • Run the second pass using the computed separators.
`java_inverted_index`	2	Compute inverted index in Java Hadoop – for every lowercased word from the articles, compute *(article name, ascending positions of occurrences as word indices)* pairs. The output should be a file with one word on a line in the following format: `word \t articleName \t spaceSeparatedOccurrences...` You will get 2 additional points if the articles will be numbered using consecutive integers. In that case, the output is ascending *(article id, ascending positions of occurrences as word indices)* pairs, together with a file containing list of articles representing this mapping (the article on line i is the article with id i).
`java_no_references`	3	An article A is said to reference article B, if it contains B as a token (ignoring case). Run a Java Hadoop job which counts for each article how many references there exists for the given article (summing all references in a single article). You will get one extra point if the result is sorted by the number of references (you are allowed to use 1 reducer in the sorting phase).
`java_wordsim_index`	4	In order to implement word similarity search, compute for each form with at least three occurrences all *contexts* in which it occurs, including their number of occurrences. List the contexts in ascending order. Given N (either 1, 2, 3 or 4), the *context* of a form occurrence is N forms preceding this occurrence and N forms following this occurrence (ignore sentence boundaries, use empty words when article boundaries are reached). The output should be a file with one form on a line in the following format: `form \t context \t counts...`

<table>
<tr><th>Task</th><th>Points</th><th>Description</th></tr>
<tr><td><code>java_wordsim_find</code></td><td>4</td><td>Let S be given natural number. Using the index created in <code>java_wordsim_index</code>, find for each form S most similar forms. The similarity of two forms is computed using cosine similarity as $\frac{C_A \cdot C_B}{|C_A| \cdot |C_B|}$, where C_F is a vector of occurrences of form F contexts.
The output should be a file with one form on a line in the following format:
<code>form \t most similar form \t cosine similarity...</code></td></tr>
<tr><td><code>java_kmeans</code></td><td>6</td><td>Implement K-means clustering algorithm as described on <code>http://en.wikipedia.org/wiki/K-means_clustering#Standard_algorithm</code>.
The user specifies number of iterations and the program run specified number of K-means clustering algorithm iterations.
You can use the following training data. Each line contains space separated coordinates of one points. The coordinates in one input naturally have the same dimension.
<table>
<tr><th>HDFS path</th><th>Points</th><th>Dimension</th><th>Clusters</th></tr>
<tr><td><code>/data/points-txt/small</code></td><td>10000</td><td>50</td><td>50</td></tr>
<tr><td><code>/data/points-txt/medium</code></td><td>100000</td><td>100</td><td>100</td></tr>
<tr><td><code>/data/points-txt/large</code></td><td>500000</td><td>200</td><td>200</td></tr>
</table>
You will get 2 additional points if the algorithm stops when none of the centroid positions change more than given ε.</td></tr>
</table>