Report on running Oozie Workflow for lab2-sort-wordcount

- **Workflow Description:**
  A workflow named *"sorted-word-count"* is wrapped in the workflow-app entity. A start node *i.e., "wordcount-node"* has been described which tells Oozie which workflow should be run first. Further, the "*wordcount-node"* action node defines a map-reduce action which is a standard Java MapReduce Job. Within the action, the job's properties have been defined, details of which is described in the further subsections. Up next, it is specified what the workflow must do if the action ends successfully and what to do if it fails. For this example, if the job is successful it goes to another action node named "*sort-node*" and if it fails, it goes to kill node named "*fail".*

  Similarly, workflow proceeds for the "*sort-node*" and if the actions turn out to be successful, it goes to the end node or else it will again go to the kill node.

- **Job description for each MR job contained in lab2-sort-wordcount Oozie workflow:**
  The first map reduce job that has been defined at the first action node "*wordcount-node*" has its configurations and properties defined in it. For instance, the mapper class for the wordcount map reduce job is defined to be WordMapper with output format for key and value as Text and IntWritable respectively. The reducer class is defined to be SumReducer.

  Now, once the wordcount mapreduce job action happens to succeed the next action sort mapreduce job is performed. And like wordcount, job properties for the "*sort-node*" is also defined. The mapper class for the sort mapreduce job is defined to be SortWordCountMapper using Hadoop api KeyValueTextInputFormat and comparator class LongWritableDecreasingComparator. The output format for key and value is defined as LongWritable and Text respectively. The reducer class for this MR job is defined to be using IndentityReducer.

- **Test Data and Result:**
  In this example, Oozie workflow is employed to perform more than MapReduce job one after the other. The first MapReduce job performs wordcount and output of the same is being used as the input of the next MapReduce job i.e., SortWordCount. This MR job sorts the words alphabetically and uses identity reducer to reduce the mapped key value pairs.
  And finally, the output is sorted word count and can be accessed from command line at /lab2-sort-wordcount/output/part-00000 and /lab2-sort-wordcount/wc-output/part-00000 in HDFS.

Report on Exploring a Secondary Sort Example

- **Introduction:**
  In Secondary Sort example, the program inputs the line in the form of three fields i.e., last name, first name and birthdate. Using map reduce and implementing various interfaces such as *WritableComparable*, the MapReduce program should output youngest person with each last name.

- **Running as a map-only job:**
  In this example, the mapper format is defined to be *StringPairWritable* type thus mapping key which is a pair of strings i.e., last name and birthyear. Since, this MapReduce program was supposed to be executed progressively while reaching to the desired output with each iteration, the first attempt was done using map only job by setting number of reduce tasks to 0. The output found after running Map job was string pair of last name and birthyear.

- **Running while using default Partitioner and Comparators:**
  While running the same job again, but this time with a number reduce tasks as 2 resulted in an output which was distributed between two files. Also in each of the files the output was sorted by last name in ascending order and year in ascending order as well. But it wasn't sorted between files and that was because the string key pair with same last name went to different reducers.

- **Running while using custom partitioners:**
  Upon adding *NameYearPartitioner* Class from command line as a parameter, the output was rectified with all records with same last name went to the same reducer, thus to the same files. But, both the last name and birth year were sorted in ascending order and not descending as desired.

- **Running while using custom sort comparator:**
  Upon adding another parameter that sets the key custom comparator class in the command line, the output found to sorted with last name(ascending) and birth year(descending). This output now seems to be sorted like the desired output.

- **Running with defined reducer class "*NameYearReducer*" instead of Identity Reducer:**
  Instead of using Identity Reducer which writes the k-v pair it receives the way it is, *NameYearReducer* class was used. This reducer class has method defined which when called performs the task on the set of values passed on by the mapper and emits only the first of its values.

- **Running while using custom group comparator:**
  Upon adding another parameter that sets the *NameComparator* class which is defined to compare two string pairs while ignoring the year field. String pairs with same last name will be

grouped into the same reduce call irrespective of the year. The output after running this MR job for final time is found to have included youngest person for each last name.