CIS 4930 Introduction to Hadoop and Big Data

Lab 8 Homework Report

Akshat Shah

U10348398

1. For project a), report should describe clearly on the following topics:
   1. lab2-sort-wordcount Oozie workflow description

* The workflow app is named as “sorted-word-count”
* The first action is named “wordcount-node” which is a MapReduce job
* If there is no error the workflow goes to “sort-node”
* If there is an error the program outputs a error message and exits
* The ouput from the first action is the input to the second action
* The second action is named “sort-node”
* sort-node is a MapReduce job
* If there is no error the program ends
* If there is an error the program outputs an error message and exits
  1. Job description for each mapreduce job contained in lab2-sort-wordcount oozie workflow
     1. wordcount-node
* The start node tells oozie to start by transitioning to wordcount-node
* The wordcount-node action defines a MapReduce action which counts the wordcount for each key value
* The properties describe what happens in the mapper and the reducer
* In the wordcount-node the mapper is defined as WordMapper whose ouput key is of the type text and output value is of the type IntWritable
* The reducer is defined as SumReducer
* The input directory is defined as inputDir and the output directory for this action is defined as wordCountOutputDir
* If this action is performed fine the output is sent to the sort-node.
  + 1. Sort-node
* The sort-node action sorts the output to the wordcount-node
* The mapper is defined as SortWordCountMapper
* The input format is defined as KeyValueTextIInputFormat
* The Output Key Comparator class is defined as LongWritableDecreasingComparator
* The map output key is of Longwritable type
* The map output value is of text type
* The reducer is just a IdentityReducer
* The input directory is wordCountOuput
* The ouput directory is outputDir
* If the sort-node works fine the program ends else there is an error message and the program is killed.
  1. Test data and results.
* The test data used was the input given with the exercise
* The output was as expected with the wordcount of each words in descending order

1. For project b), report should describe clearly on the following topics:
   1. Lab introduction.

* In this lab we run a MapReduce job in multiple ways and see the effects of various components in a secondary sort program.
* The program accepts input line of the form

*Lastname firstname birthdate*

* The goal of the sort is to identify the youngest person with each last name.
* We run the program with
  + - * 1. Run using Map-only job
        2. Run using the default comparator and partitioner but set the number of reduce tasks to 2
        3. Run using custom partitioner
        4. Run using the custom sort comparator
        5. Run with the NameYearReducer
        6. Run with the custom group comparator
* After each and every step we move closer to the desired outputs and can see how each step affects the output.
  1. Lab subsection: Run using the default Partitioner and Comparator: Description on the default partitioner and comparators and check the job output with the default partitioner

*The job can be run by using the command*

*hadoop jar secsort.jar example.NameYearDriver\*

*-Dmapred.reduce.tasks=2 nameyeartestdata secsortout1*

* This sets the number of reduce tasks to 2 and outputs the data to secsortout1.
* The output to this task is sorted both by last name and year and both are sorted in ascending order.
  1. Lab subsection: Run using the custom Partitioner: Description on the custom partitioner: NameYearPartitioner, check the job output with the custom partitioner, and compare the results with those generated with default partitioner.

*The command to run the custom partitioner is*

*hadoop jar secsort.jar example.NameYearDriver \*

*-D mapred.reduce.tasks=2 \*

*-D mapreduce.partitioner.class=example.NameYearPartitioner nameyeartestdata secsortout2*

* The custom partitioner works to partition the data from the first part (last name) of the string such that all the keys with the same last name go to the same reducer.
* The results from the default partitioner had the same last names repeated in outputs from both reducers.
* While, the outputs from the custom partitioner made sure that all the keys with the same last name go to the same reducer.
  1. Lab subsection: Run using the custom sort comparator: Description the custom sort comparator, check the job output with the custom sort comparator, and compare the results with those generated with the custom partitioner

*The command to run the custom sort comparator is*

*hadoop jar secsort.jar example.NameYearDriver \*

*-D mapred.reduce.tasks=2 \*

*-D mapred.output.key.comparator.class=example.NameYearComparator \*

*-D mapreduce.partitioner.class=example.NameYearPartitioner \*

*nameyeartestdata secsortout3*

* The outputs for the custom comparator compare the Name/Year pairs.
* If the names are equal the comparator compares the years and outputs the years in a descending order.
* Thus, the names are output in an ascending order and the years are output in descending order.
* While in default comparator the name and year are both in ascending order.
  1. Lab subsection: Run with the NameYearReducer: Description on the NameYearReducer: Description on reducer: input key, value types, output key, value types, and method manipulation on the input key value pairs. Run the job with NameYearReducer, check with the results.

*The command to run the NameYearReducer is*

*hadoop jar secsort.jar example.NameYearDriver \*

*-D mapred.reduce.tasks=2 \*

*-D mapreduce.partitioner.class=example.NameYearPartitioner\*

*-D mapred.output.key.comparator.class=example.NameYearComparator \*

*-D mapreduce.reduce.class=example.NameYearReducer \*

*nameyeartestdata secsortout4*

* The NameYearReducer has the input key type StringPairWritable
* The NameYearReducer has the output key type text.
* The NameYearReducer has the output key type text.
* The NameYearReducer has the output key type text.
* The NameYearReducer only emits the first set of values for each set of values passed to the reducer.
* It is done by using iterator().next() which iterates to next set of values.
* The output for NameYearReducer outputs multiple values for each last name but different years because the reducer is being grouped by the full key with the name and the year.
* Also, the keys were not output because the reducer does not include them in the context object.
* Thus, the only way a record is eliminated is if two people have same last name and the same year.
  1. Lab subsection: Run using the custom group comparator: Description the custom group comparator. Run the job and check with the results.

*The command is*

*hadoop jar secsort.jar example.NameYearDriver \*

*-D mapred.reduce.tasks=2 \*

*-D mapreduce.partitioner.class=example.NameYearPartitioner \*

*-D mapred.output.key.comparator.class=example.NameYearComparator \*

*-D mapreduce.reduce.class=example.NameYearReducer \*

*-D mapred.output.value.groupfn.class=example.NameComparator \*

*nameyeartestdata secsortout5*

* The NameComparator compares keys by only comparing the Last Name and not the year.
* Thus, the pairs with same last name will be grouped together is the same reduce call.
* The results were as expected where the output only contains the youngest person for each last name and no keys are output.