Mandar Darwatkar (861141010)
Karishma Dash (861188164)

CS236 Project
University of California, Riverside
March 19, 2015 - Winter 2015

README

Username: mdarw001, kdash001
Node: z3

User Guide:

1. Ensure run.sh has execute permission.
2. Ensure that run.sh, WeatherProject.jar and lib directory are contained in same directory.
3. Run run.sh script.
4. Input absolute directory path of temperature dataset when prompted.
   e.g. /user/sjaco002/input
5. Input absolute directory path of *WeatherStationLocations.csv* directory when prompted.
6. Input desired output directory when prompted.

We modularize the problem according to the sub-problems to attain simplicity, reusability and scalability:

Job 1 – Read & parse the temperature records from user specified location and extract required columns (We also extract precipitation data).
Job 2 – Read & parse *WeatherStationLocations.csv* from user specified location and extract state-station pairs. Classify valid unclassified data.
Job 3 – Inner join record data with states.
Job 4 – Calculating temperature variation for merged data.
Job 5 – Sort the final output and write to user specified directory.

We describe each Job in following:

Job 1 – READ & PARSE

In this job we read and parse the records data. We extract STN, YEARMODA, TEMP and PRCP. As the precipitation is computed over varying period, we calculate average precipitation per hour. This is calculated by dividing the values by corresponding hours supplied as tail label. Data is sorted on station ID which will be later used for join. Sort is implicitly handled by Hadoop with requirement if setting key to sort key.

CPU Time: ~35sec

Job 2 – TAGGING UNCLASSIFIED DATA

Apart from reading and parsing the *WeatherStationLocations.csv,* we go ahead and tag unclassified data (stations). Data is sorted on station ID which will be later used in join phase. Mapper does two major tasks:

1. Cleaning & Parsing:

   While reading the file, we discard header and non-USA data. The dataset contains locations that are tagged with country USA and not tagged with any state, but have coordinates outside continent e.g. 0^0 latitude, 0^0 longitude, empty latitude-longitude, etc. We discard such entries during parsing. Also, there are some stations that are associated with multiple states. Since we do not have any information in records dataset, such deficiency can be handled with following solutions:

   - Associating records from second dataset to each matching station-state entry.
   - Eliminating such duplicates and then associating records from second dataset to only single station-state entry.
   - Mapping all such stations to any one of the states and continue processing.

   We take up third approach by using HashMap to maintain unique state-station pair.

2. Tagging unclassified data:

   Assumption: USA boundary latitudes and longitudes are already available.
   For this project, these are obtained from Google Maps. These coordinates enclose USA mainland only, while excluding Alaska and other islands.

   To tag unclassified data we do the following,

   - First, if the record has non-empty state, we extract state-station pair. Simultaneously, we maintain a list of average latitude-longitude value for each state. This gives a benchmark that will be used later to classify unclassified records.
   - Second, the untagged data is classified into two new classes – AT: Atlantic region and PC: Pacific region, based on their latitudes and longitude relative to the boundaries we have. Any state which has latitude within the USA boundary and longitude outside of the boundary is classified into Atlantic and Pacific.
   - Remaining untagged data is classified into appropriate states using nearest-neighbor. We implement nearest neighbor by calculating the relative Euclidean distance between a station to the average latitude longitude values calculated for each state and map the station to the state with minimum distance between them.
   - Finally we have all valid data which has been tagged and supplied to next phase.

   We succeed in classifying ~13,800 valid untagged data. We use brute-force approach to verify our results.

CPU Time: ~7sec

Job 3 – JOIN AND AGGREGATE

This job handles joining the two datasets. We leverage sorting, partition and join feature provided by Hadoop to gain deeper understanding into Hadoop functionality, instead of HashMap lookup or other merge-join algorithm.

We use Map-side joins to take advantage of:

- Data already sorted on join key i.e. station ID.
- One dataset (WeatherStationLocations.csv) can fit in memory.

We get the sorted data from previous job without any additional sorting overhead as Hadoop sorts the data on key value. Following is the brief description of components involved:

- `CompositeInputFormat` which merges the records before they reach to mapper.
- Using `mapreduce.input.keyvaluelinerecordreader.key.value.separator`, we specify the character that separates the key from value. This helps identify the join key.
- This probelm requires inner join. This join expression is specified in `CompositeInputFormat.compose`
- `KeyValueTextInputFormat` will use the separator character to find the join key.
- Finally, in `CombineValuesMapper` we append the keys to the values merged into one string. Since the join key is the first column, our mapper naturally eliminates the duplicate keys from the joined datasets.

Thus it bears repeating at this point that, considering the gains in efficiency from being able to do a map-side join, it may be worth the cost of running additional MapReduce job.

Before sending the data to next phase we average out the data-temperature and precipitation, for each state.

CPU Time: ~12sec

Job 4 – COMPUTE TEMPERATURE VARIATION

In job 4, we find month with maximum temperature and month with minimum temperature for each state. From this data we derive temperature variation for each state. We also send the average precipitation for above months.

CPU Time: ~2sec

Job 5 - SORT

Here MapReduce job does nothing more than output the data ordered by temperature variation in ascending order. We do not employ any sorting algorithm since Hadoop sorts the data based on key (temperature variation).

CPU Time: ~1sec

Output file has following structure:

Total data rows : 55 (53 + AT + PC)
Total data columns: 6 (4 + minimum month's precipitation + maximum month's precipitation)

STATE	MINIMUM	MAXIMUM	DIFFERENCE	MIN_MONTH_PRCP	MAX_MONTH_PRCP

	Base	Base + enriched data	Base + enriched data + classifying unclassified data
Execution Time	51sec	1min 2sec	1min 20sec