Assignment 7

Map-Combine-Reduce

This assignment is on map-combine-reduce, which is a distributed and scalable way of extracting/mining required information from multiple datasets stored on multiple servers. Follow the tutorial to understand how you can design mapper and reducer for specific queries/operations.

Tutorial on combiner:

We learned about map-reduce in the last week. This week we are looking into combiners too.

In many real world scenarios, the mapper has to send too many key-value pairs, which then reach the Hadoop/MapReduce management system to be grouped and sorted. Having too many key-value pairs can become a bottleneck for the system. Instead if we introduce some form of grouping in the mapping phase itself, then the load on the whole system will be significantly reduced. So, we plan to use combiners (mini-reducers at local mapping level wherever possible) in the mapping phase before passing the key-value pairs to be shuffled and sorted.

Combiner Reference:

https://www.tutorialspoint.com/map_reduce/map_reduce_combiners.htm

Dataset:

We will be using a similar dataset as that of employee-department association of Assignment 6. The data contains 10 files containing logs of email communication for 10 different events of the company. You have to extract required information from this data. Below is a sample of data available in the file.

(event1.txt)

```
5 8
7 9
9 6
6 5
10 2
2 11
6 5
7 9
```

Here each line in the file named “event1.txt” represents an email communication between two departments for the Event 1. Ordering of the logs doesn’t matter here. This time the network will be undirected. Note that a node-pair can occur multiple times in the file.

Queries:

The queries are to be implemented in the mapper (also combiners wherever asked) and reducer. You need to implement the following queries in this assignment.

1. You have been given the dataset of logs of email communication for each event between the departments of the company. For each department, you want to find the number of distinct departments (excluding the department) with whom emails have been communicated by the respective department. You have to group the departments by the number of email communications with other departments and for each group count how many departments belong to the group. [30]

   Note- As there are 10 different files, you will have to write mapper, combiner, and reducer routines. You are allowed to use multiple mappers and reducers. The combiner must behave as a mini-reducer at mapper level; i.e. each event’s file should go through mapper and combiner individually. The combiner shouldn’t just be receiving and sending the same key-value pairs.

   Output format- The result will have lines such as: 15 30 which means that there are 30 departments each who have email communication with 15 different departments. The output file should contain one such count in each line.

Deliverables: mapper1.py, combiner1.py, reducer1.py, result_event.txt. Note: You may have to use mapper2.py, reducer2.py, etc. depending on your logic. Please include those files as well if needed.

2. We want to find the strongly connected nodes (i.e., departments) using how many times a pair of nodes is involved in email communications in the 10 events. If a pair of nodes has at least 50 email communications in all the events in total, then you have to consider them to be strongly connected and have an edge in between them. Now find all such strongly connected pairs of nodes to create the event network. [30]

   Note- As there are 10 different files, you will have to write mapper, combiner, and reducer routines. Save the network as "result_strong.txt". You have to discard self-loops when generating the network.

   Output format- Save the result_strong.txt file as an edge list format (same as that of the input files).

   Deliverables- mapper.py, combiner.py, reducer.py, result_strong.txt. There should be one mapper and combiner for each of the files.

3. You want to find out how many common departments each pair of nodes have in the event network created in the last question. If there are edges like (v1,v2), (v2,v3), then the node pair (v1,v2) has one common department, i.e., v3. Write mapper and reducer routines and save the result in "result_common.txt". [30]

   Deliverables- mapper.py, reducer.py, result_common.txt

   Output format- Node1, Node2, #commonDepartments

   Hint- If A is the adjacency matrix for the network, A^2 will have the number of common departments. The mapper will take the result_strong.txt from the last question as input.

4. **[Bonus]** You want to check whether the department with maximum number of email communication has a path to reach to the department with minimum email

communication. Given the network “event1.txt”, construct a weighted undirected network in which nodes will be the departments and edge weight will be the number of email communication that exists between the two departments. You need to discard the self-loops in the network.

Now, find the node with maximum degree (i.e., email communication) as well as the nodes with minimum degree. If there are more than one nodes having the minimum degree, consider all of them. Find the shortest path to all the minimum degree nodes from the maximum one and the corresponding shortest path length. [50]

Output format- NodeWithMaxDegree, NodeWithMinDegree, pathLength

Deliverables- mapper.py, reducer.py, result_shortest.txt

Evaluation scheme- Results- 90 marks, Coding style- 10 marks (for query1, query2 and query3)

Results- 40 marks, coding style- 10 marks (for query4)

Important Instructions

1. **The mapper routine can pass over the event(y).txt file only once. Python dictionaries should not be used in the mapper routine. However, you can use arrays or dictionaries in the reducer only. These must be limited to one dimension only. The combiner and reducer will have access to only the stream of key-value pairs but nothing else. Any violation will attract a penalty upto 100% of the marks assigned.**
2. **Submission Rule:** All deliverables must be compressed as **.tar.gz** (gzip) and named “**A7_<RollNo>.tar.gz”**. For each query, make a directory named “**Query<no.>”**. Your files must be inside the respective directories. Strictly adhere to this naming convention. Submissions not following the above guidelines will attract penalties.
3. **Makefile:** The folder **must** have it’s own Makefile to execute the routines one after another. You can find a relevant tutorial here (https://opensource.com/article/18/8/what-how-makefile).
   Note: “sort” behaves differently in different environments. Ensure that you use sort in a way which works properly on a linux machine.
4. **Code error:** If your code doesn’t run or gives error while running, you will be awarded with zero mark. Your code must run correctly on **a linux machine**.