Assignment 5

Map-Combine-Reduce

Design Lab

3rd February 2023

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, combiner and reducer for specific queries/operations.

Tutorial on combiner

We learned about map-reduce in the last assignment. In this assignment, 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 the 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

Tasks

1. Study Combiner basics.
2. Write the necessary mapper, combiner and reducer routines in python to implement the queries mentioned in later sections.
3. Ensure that the reducer routine prints the results of queries to a text file (as asked in the later sections).

Dataset

We will be using a dataset of an anonymized Web server log file from a public relations company. Please download the data from https://drive.google.com/file/d/1Bno4RCYlZhg1f3c7_JH4qZog3oFIxVDz .

In the files "**access_log1.txt**", "**access_log2.txt**", "**access_log3.txt**", "**access_log4.txt**", "**access_log5.txt**", "**access_log6.txt**", "**access_log7.txt**", "**access_log8.txt**", "**access_log9.txt**", "**access_log10.txt**", "**access_log11.txt**", "**access_log12.txt**" each line represents a hit to the Web server. It includes the IP address which accessed the site, the date and time of the access, and the name of the page which was visited.

The logfiles are in the Common Log Format:

10.223.157.186 - - [15/Jul/2009:15:50:35 -0700] "GET /assets/js/lowpro.js HTTP/1.1" 200 10469

%h %l %u %t \"%r\" %>s %b

where,

1. %h is the IP address of the client
2. %l is identity of the client, or "-" if it's unavailable
3. %u is username of the client, or "-" if it's unavailable
4. %t is the time that the server finished processing the request. The format is [day/month/year:hour:minute:second zone]
5. %r is the request line from the client is given (in double quotes). It contains the method, path, query-string, and protocol or the request.
6. %>s is the status code that the server sends back to the client. You will see see mostly status codes 200 (OK - The request has succeeded), 304 (Not Modified) and 404 (Not Found). For more info on status code - https://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html
7. %b is the size of the object returned to the client, in bytes. It will be "-" in case of status code 304.

== Sample data for the file **access_log1.txt** ==

```
10.223.157.186 - - [15/Jul/2009:14:58:59 -0700] "GET / HTTP/1.1" 403 202
10.223.157.186 - - [15/Jul/2009:15:50:35 -0700] "GET / HTTP/1.1" 200 9157
10.223.157.186 - - [15/Jul/2009:15:50:51 -0700] "GET / HTTP/1.1" 200 9157
10.223.157.186 - - [15/Jul/2009:20:50:29 -0700] "GET / HTTP/1.1" 200 9157
10.223.157.186 - - [15/Jul/2009:20:50:38 -0700] "GET / HTTP/1.1" 200 9157
10.223.157.186 - - [15/Jul/2009:21:04:42 -0700] "GET / HTTP/1.1" 200 524
10.223.157.186 - - [15/Jul/2009:21:24:10 -0700] "GET / HTTP/1.1" 200 7616
10.82.30.199 - - [29/Jul/2009:08:46:38 -0700] "GET / HTTP/1.1" 200 7616
```

==========

Part 1 - Queries

The queries are to be implemented in the mapper, combiner and reducer phases. Some of them may give empty results. You need to implement the following queries in this assignment. The queries have an imaginary backstory to help you find a real-world perspective in this assignment.

Assume you are an engineer of the company. You want to conduct a quantitative analysis of the web server log of the company using the access log files available to you. Please use MapReduce to answer the following queries.

1. For this query consider the files “**access_log1.txt**”, “**access_log2.txt**”, “**access_log3.txt**”, “**access_log4.txt**”, “**access_log5.txt**”, “**access_log6.txt**”, “**access_log7.txt**”, “**access_log8.txt**”, “**access_log9.txt**”, “**access_log10.txt**”. Find the average size (bytes) of every file type requested by the users. Files can be classified into the following types - image (.png, .jpg, .gif, .ico), video (.mp4, .flv), audio (.mp3) and webpage (.html, .css, .js, .php) [20]

   Note: The result will have lines such as: image 16.50 which means that the average size of the requested image files is 16.50 bytes. Round the result to 2 decimal points. The output file should contain one such size in each line. The ordering of the file types does not matter. In case of “-” in %b (size) consider the size to be 0 bytes.

2. For this query consider the files **‘access_log11.txt’** and **‘access_log12.txt’**. Consider the network formed by the ip-addresses and the requested resources (only GET requests). An ip-address and a resource is considered connected then if and only if that resource was requested by the ip-address. Now, this connection among the ip-addresses and the resources form a network. Find the total number of unique pairs of ip-addresses which are disconnected from each other.

   [30]

   Note: The output file should contain a single integer, the required number of pairs in one line. In case of “-” in %b (size) consider the size to be 0 bytes.

Deliverables: 2 sets of python codes (mapper, combiner and reducer), 2 Makefiles, 2 result.txt, and 2 readme files explaining your approach.

Part 2 - MapReduce with Multiprocessing

You need to implement the above 2 queries in MapReduce using multiprocessing. Consider these as Query3 and Query4 respectively.

[20 + 20 = 40]

Note: To sort the output of the mapper and the combiner, you can execute the Linux command “sort” from python code directly on the output files. Don’t sort the data structures of the mapper and the reducer directly, but the corresponding output files.

Deliverables: 2 python codes (query3.py and query4.py), 2 result.txt, 2 readme files explaining your approach, and 2 pdf files showing the comparison between the time taken in Query 1 vs Query 3, and Query 2 vs Query 4.

Evaluation Scheme: Results: 90 marks, Coding Style: 10 marks

Important Instructions

1. **The mapper routine can pass over the access_log(y).txt file only once. You cannot use any intermediate file structure or data structure to save data for Query 1 and Query 2. Python dictionaries should not be used in the mapper routine. However, you can use one-dimensional arrays in the mapper. You can use multi-dimensional arrays and multi-dimensional dictionaries in the combiner and the reducer. The combiner and reducer can't access the data file directly. So, the mapper should just go through the data and generate a series of key-value pairs which will then be passed to the suitable sort command. Finally, the combiner and then the reducer should just go through the stream of tokens once to give the required output. You cannot use the NetworkX library. You cannot perform any kind of sorting either in the mapper or the reducer. You can only sort the output of the mapper using the Linux sort command before sending it to the combiner and the reducer. Any violation will attract a penalty of up to 100% of the marks assigned.**

2. **Submission Rule:** All deliverables for the above functionalities must be compressed as **.zip** and named **"<RollNo>.zip"**. For each query, make a directory named **"Query<no.>"**. Your files must be inside the respective directories. Strictly adhere to this naming convention. You must prepare a report comparing the performance after parallelisation. Submissions not following the above guidelines will attract penalties.

3. **Makefile:** Each query folder, Query 1 and Query 2 **must** have it's own Makefile to execute the routines. 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**.