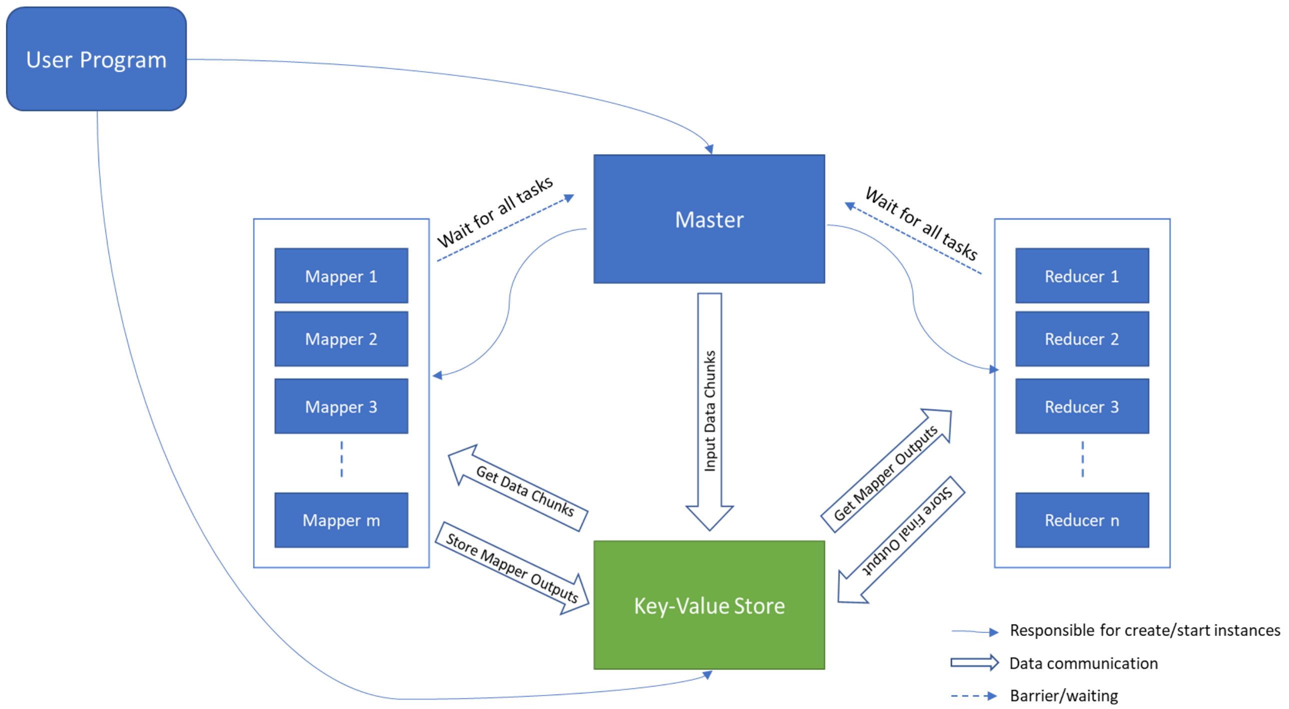
Assignment 3 – Distributed MapReduce on GCP

Report – 10/23/2020

In this assignment, designed and implemented a distributed MapReduce System from scratch using python programming and has been deployed on the google cloud platform. This version of map-reduce framework has been designed in such a way that, any map-reduce function can be run through this framework, given the function is well defined inside the mappers and reducers. For now, this framework assumes that user is only looking to get word-count and inverted index. Communication between nodes is done using RPC based interface. To explain briefly about MapReduce and GCP. MapReduce is a programming model designed for processing large volumes of data in parallel by dividing the work into a set of independent tasks. Google Cloud Platform (GCP) is a suite of cloud computing services that runs on the same infrastructure as Google.

Design Details:

The Master instance accepts the input from user program and performs the user-provided operation. Mapper & reducer instances perform mapping & reducing tasks respectively. Key value store instance stores the intermediate & final data. There is a proper synchronization between each instance. Master is the backbone of the system, which co-ordinates with all other entities.



1. User Program (*user_program.py*): This is the main program that creates master instance and starts master server and similarly kvserver instance and starts key value server. This program instructs master to run init_cluster which creates n mappers and m reducers which are given as input in config.ini. Then this program instructs master to start mapper and reducer tasks and finally this program destroys the cluster created.

2. Key Value Server (*KVServer.py*): This sever is responsible to set and get data from file system for master or any of the workers. All the data is stored in text format by this key value store. Input data comes as chunks to this server and it stores, and in case of inverted index it stores a dictionary mapping of each mapper input with the document file name which is saved in file_mapping.txt. Mappers take input data chunks and respond with their results to this server to save as inputs for reducer. Then, when reducers start, they read their corresponding inputs and write their final output to final_kv_store.txt file.

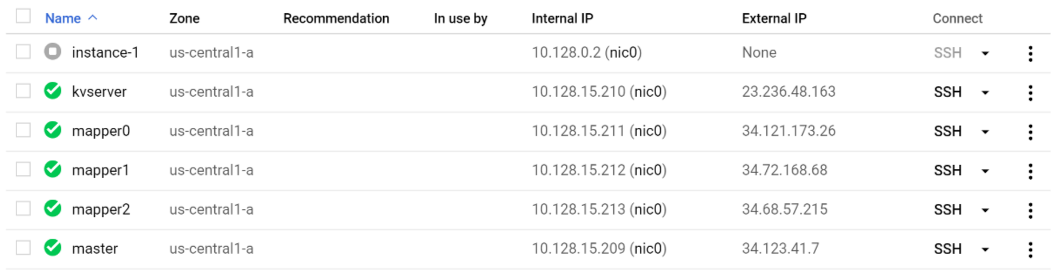
3. Master Server (*master.py*): In a literal sense, this server behaves as the master for this system. It is in charge of all the communication taking place between all the servers. This starts all the required mapper and reducer instances as per the instruction from user program. In run mapreduce step, it coordinates with mappers, reducers by providing which data they have to work from key-value storage. And finally this server destroys all vm instances created for each worker.

4. Mapper (mapper_client.py): The mapper (say N in number) instances are invoked by the Master instance, and then they establish a connection with the Master Instance using RPC calls. After being connected to the Master, the mapper instances parallely start accepting input from the Master Instance, in the form of which input chunk to read from Key Value store to process. At the master instance the input data in the form of a large document which is equally divided into N chunks by the master instance. These mapper instances then concurrently send data to the Key-Value Store Instance. These mapper outputs which will be input to reducers are cleverly created so that each reducer gets to read only one of these files. In this way, data is already grouped according to the reducers(thru consistent hashing) into different files in the key value store, and that way every reducer fetches relevant data to it. Hash(word) mod n distributes mapper outputs into 'n' groups for 'n' reducers.

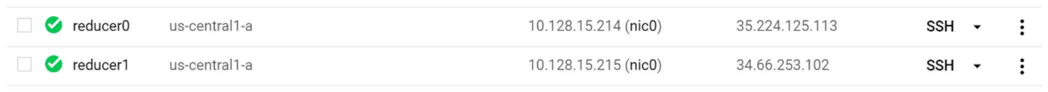
5. Reducer (reducer_client.py): The reducer (say M in number) instances are invoked by the Master Server, and then they establish a connection with the Master Instance using RPC calls. After being connected to the Master, the reducer instances parallely start accepting input from the Master Instance, in the form of which mapper output to read from Key Value store to process. These reducers read their corresponding data, do their functionality and write the final output concurrently to one single file.

Implementation Details:

- User programs starts master and key value server instances and runs their respective python scripts
- All the necessary files and scripts are placed in a sample instance and image of it is created. This image is used to create all worker instances but runs python script as per their role (mapper/reducer).
- *gcp_instance.py* is a instance handler script which will utilize google cloud APIs to create, start, stop, destroy instances within a given project and specified image by the config file.
- When starting an instance, a script is run to keep the workers ready to listen from master about the task etc.
- After the init_cluster instruction comes to master , it creates/starts first mappers and then reducers



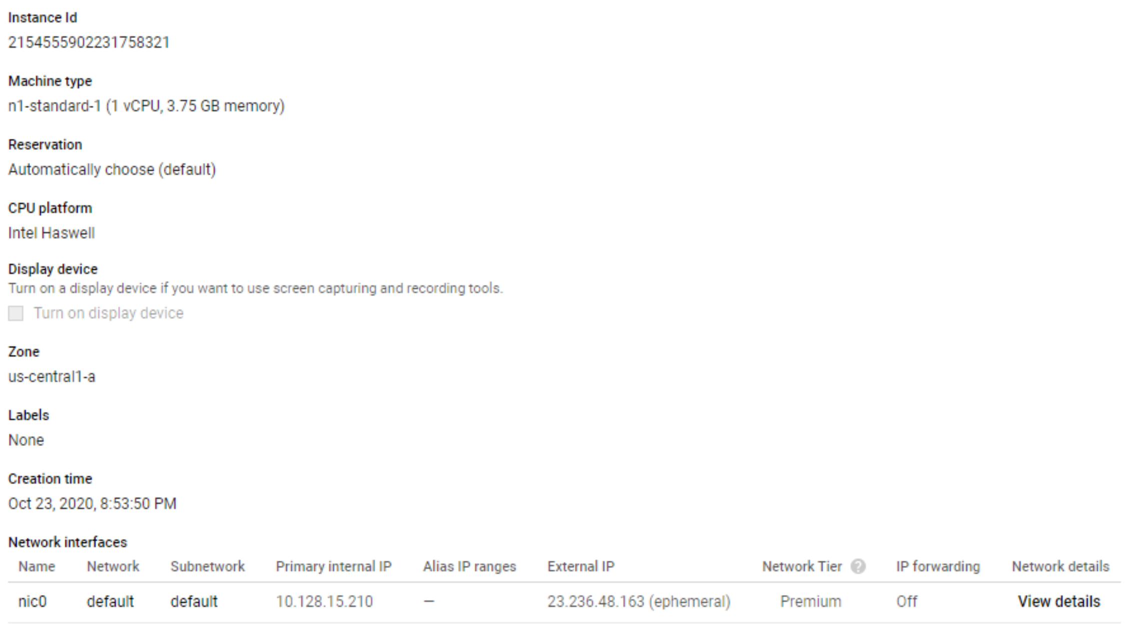
Name	Zone	Recommendation	In use by	Internal IP	External IP	Connect
○ instance-1	us-central1-a			10.128.0.2 (nic0)	None	SSH
✓ kvserver	us-central1-a			10.128.15.210 (nic0)	23.236.48.163	SSH
✓ mapper0	us-central1-a			10.128.15.211 (nic0)	34.121.173.26	SSH
✓ mapper1	us-central1-a			10.128.15.212 (nic0)	34.72.168.68	SSH
✓ mapper2	us-central1-a			10.128.15.213 (nic0)	34.68.57.215	SSH
✓ master	us-central1-a			10.128.15.209 (nic0)	34.123.41.7	SSH



✓ reducer0	us-central1-a	10.128.15.214 (nic0)	35.224.125.113	SSH
✓ reducer1	us-central1-a	10.128.15.215 (nic0)	34.66.253.102	SSH

- Each of this start/create will wait for the instance to be ready before running their respective python scripts
- When all workers are ready, and run map reduce instruction comes from user program, then master assigns data to workers to run their tasks and workers then output to key value server.
- Suppose the mapper or reducer functions are not supported, the master will respond to user that this it not the valid mapper or reducer function at the moment
  Ex: *Not a valid application*
- The final output is stored in key value store
- After done with all the task, all workers are terminated.
- In case of any failures of any mappers or reducers instances are not destroyed, they are automatically terminated at the end.

VM Details:



Instance Id
2154555902231758321

Machine type
n1-standard-1 (1 vCPU, 3.75 GB memory)

Reservation
Automatically choose (default)

CPU platform
Intel Haswell

Display device
Turn on a display device if you want to use screen capturing and recording tools.
Turn on display device

Zone
us-central1-a

Labels
None

Creation time
Oct 23, 2020, 8:53:50 PM

Network interfaces

Name	Network	Subnetwork	Primary internal IP	Alias IP ranges	External IP	Network Tier	IP forwarding	Network details
nic0	default	default	10.128.15.210	–	23.236.48.163 (ephemeral)	Premium	Off	View details

Cost of running experiments:

- Cost of running these experiments are very low as we are terminating node instances after all the tasks are done
- Initial cost credit was $50. Credits remaining after thoroughly testing MapReduce architecture is around $47

Assumptions:

- Works only for word count and inverted index
- For inverted index – number of mappers should be equal to number documents in the directory
- For inverted index – all documents should be test files only
- Config file and all other required files are in the image with which the instances are created

Experiments/Performance:

- With 3 mappers and 2 reducers
  - Word count on 'book.txt' of 8kb
    - Time taken till creating/starting cluster is 200.09443163871765s
    - Time taken for the whole process is 203.91879320144653s
    - Effective time taken by MapReduce tasks is 3.82s
  - Word count on 'short_stories.txt' of 543kb
    - Time taken till creating cluster is 181.31152629852295s
    - Time taken for the whole process is 205.43445944786072s

        - Effective time taken by MapReduce tasks is 24.12s
    - Inverted Index on 3 'books' of average 10kb
        - Time taken till creating cluster is 198.8377766609192s
        - Time taken for the whole process is 203.48830389976501s
        - Effective time taken by MapReduce tasks is 4.65s
- For a test file of 543 kb which is named as 'short_stories.txt'
    - Word count application is run with
        - 5 mappers and 5 reducers is – Effective time taken by this MapReduce task is 10.6 seconds
        - 3 mapper and 2 reducer – Effective time taken by this MapReduce task is 24.12 seconds
- One of the test case: For inverted index on duplicate text files gives same result for each word, which shows that key value store is handling all the data properly
    - Sample output for this case:

```
('it', 'book1|15 book2|15 book3|15')
('have', 'book1|14 book2|14 book3|14')
('been', 'book1|4 book2|4 book3|4')
('in', 'book1|33 book2|33 book3|33')
('vain', 'book1|1 book2|1 book3|1')
('for', 'book1|5 book2|5 book3|5')
('to', 'book1|36 book2|36 book3|36')
('the', 'book1|63 book2|63 book3|63')
('weather', 'book1|1 book2|1 book3|1')
('and', 'book1|55 book2|55 book3|55')
('were', 'book1|11 book2|11 book3|11')
('was', 'book1|23 book2|23 book3|23')
```

Test Cases:

- No input file passed:
  Master response - `ERROR: Data not provided or not in the format expected`
- Unsupported map function passed:
  Master response - `ERROR: Not a valid application`
- Unsupported reduce function passed:
  Master response - `ERROR: Not a valid application`

Implementation Improvements:

- Handling fault tolerance – when a mapper or reducer fails
    - In original implementation these are handles by
        - Running the failed tasks again
        - Having backup tasks
- Overall MapReduce framework could have been made more interactive, for example a webpage that takes all the inputs and then performs computations
- A bunch of different map and reduce functions can be implemented

List of Files:

1. Input files/folder – book.txt, short_stories.txt, books(directory for inverted index)
2. Code files – gcp_instance.py, user_program.py, KVServer.py, master.py, mapper_client.py, reducer_client.py, wordcount_mapper.py, wordcount_reducer.py, invertedindex_mapper.py, invertedindex_reducer.py
3. Startup scripts - kv_startup_script.sh, master_startup_script.sh, reducer_startup_script.sh, mapper_startup_script.sh
4. Config file – config.ini
5. Log files – master_log.txt, gcp_mapreduce_wc_downloaded-logs-20201023-180242.json
6. Output files - final_kv_store_ii_3m2r.txt, final_kv_store_wc_3m2r.txt

How to run the code?

From any machine run

➢ *python3 user_program.py*

- Only the IP address of workers is hard coded.
- Along with the above list of files, json key of the project is required to run this code