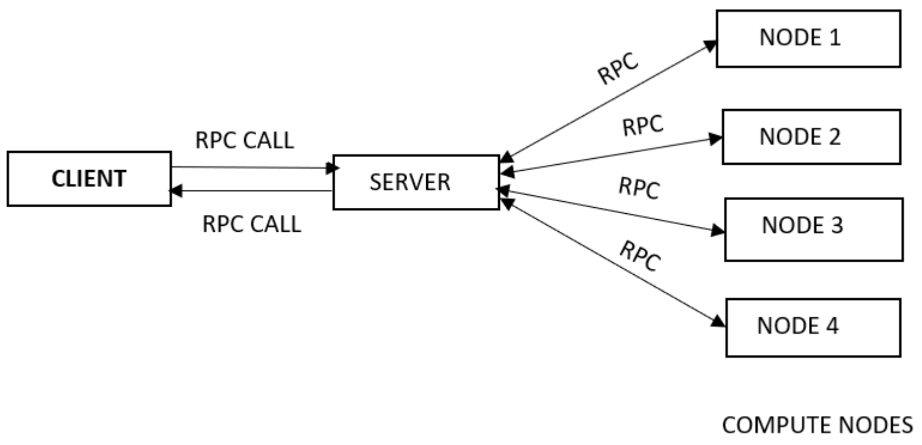
INTRODUCTION TO DISTRIBUTED SYSTEMS (CSCI 5105)
PROGRAMMING ASSIGNMENT-1

DESIGN DOCUMENT

GROUP MEMBERS:
Sai Pratyusha Attanti *(attan005@umn.edu, Student ID: 5656785)*
Navya Ganta *(ganta016@umn.edu, Student ID: 5673223)*

SYSTEM DESIGN:



This system can perform a simple image processing task edge detection on large datasets by splitting the jobs and assigning them to the compute nodes that perform computations. And all the communications between the nodes are synchronous. The client sends the job to the server with the information about the path of the shared input directory. And after the server receives the job it splits the job into tasks and assigns them to a compute node for edge detection which can perform multithreading image processing on each image.

Task is assigned to the compute nodes based on the scheduling policy given in the config file. The two scheduling policies we have implemented are RANDOM and LOAD BALANCING. And based on the load probabilities assigned to each node the tasks are scheduled accordingly.

client.py:
This file contains the implementation of our client. The client makes RPC call to the server with the location of the images the server has to perform image processing on. And it will wait till the server completes the job and once the server notifies the client about the job completion by sending the file name which contains the data about the time taken by the job the client reads the file and prints the time elapsed for the job.

server.py:
This file contains the implementation of our threaded server (Tthreadedserver). Once the server receives the RPC call from the client with the information about the job. The server splits the job into images, assigns the task to compute node randomly. If the compute node rejects the task, the server assigns it to another node. The server logs the status and time taken by each task into a

text file from the information received through the RPC calls from compute nodes after the task completion. When all tasks are done, it notifies and returns the results to the client.

computeNode.py:
This file contains the implementation of our threaded compute nodes(Tthreadedserver). We have implemented 4 compute nodes for this project. Once the compute node receives the task from the server, If the scheduling policy mentioned in the config is 'load_balancing', the compute node may either accept or reject the task based on the load probability of that node. Each thread is created for each accepted task and delays are injected into the task according to the load probability of that node irrespective of the scheduling policy mentioned in the config file. After completion of each task, it saves the resultant image in output_dir, and also makes an RPC call to the server, to inform about the stats of the task.

CONFIGURATION :
This config.cfg file stores the necessary information required to configure the system dynamically.

- Port numbers of the client, server and compute nodes.
- Path to the input and output directories(input_dir and out_dir).
- Scheduling policy - 'random' and 'load_balancing'.
- Load probabilities assigned to each of the compute nodes. (default load probabilities : 0.2, 0.2, 0.2, 0.2)
- Delay injected to the task. (default delay : 3sec)

DATA:
The sample input data of the image processing task are in the input_dir folder, and the outputs of these tasks will be saved in the output_dir folder. Both these folders are present inside the proj_dir folder.

INSTRUCTIONS:
To run this project we will need Python 3 and Thrift.

- Make sure to modify the env.txt file according to the location of libraries in your system.
- Machine.txt has the information about the machines on which the different nodes are expected to execute. If needed this can be changed.
- Check the config.cfg file to make sure all the ports and other options are as desired.
- Run grading.sh script.
  Or
  Run 'python3 computeNode.py 0' to run the first node.
  Run 'python3 computeNode.py 1' to run the second node.
  Run 'python3 computeNode.py 2' to run the third node.
  Run 'python3 computeNode.py 3' to run the fourth node.
  Run 'python3 sever.py' to run the server.
  Run 'python3 client.py' to run the client.

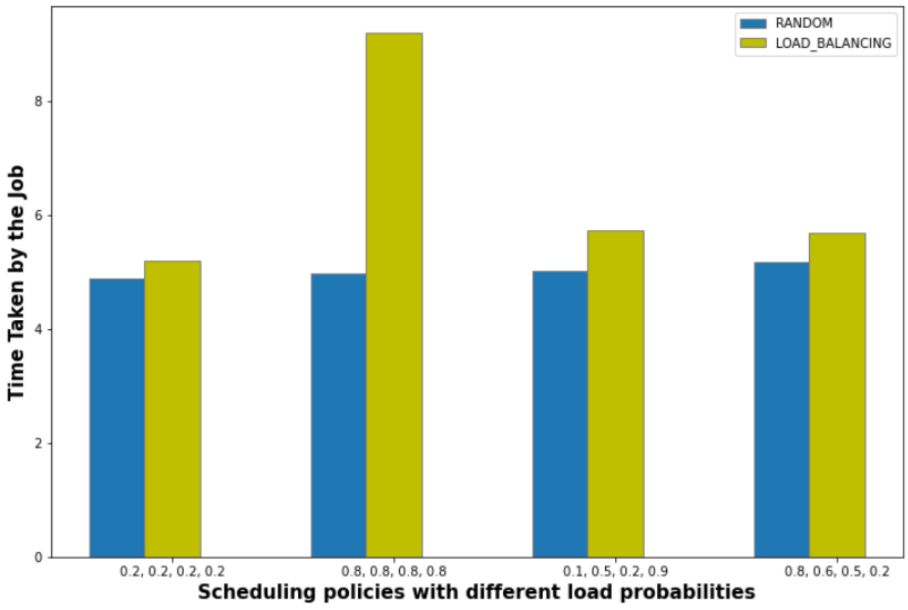
PERFORMANCE ANALYSIS:

We have tested the performance of our system with different data sets, scheduling policies, and load probabilities. Here are the results of our analysis.

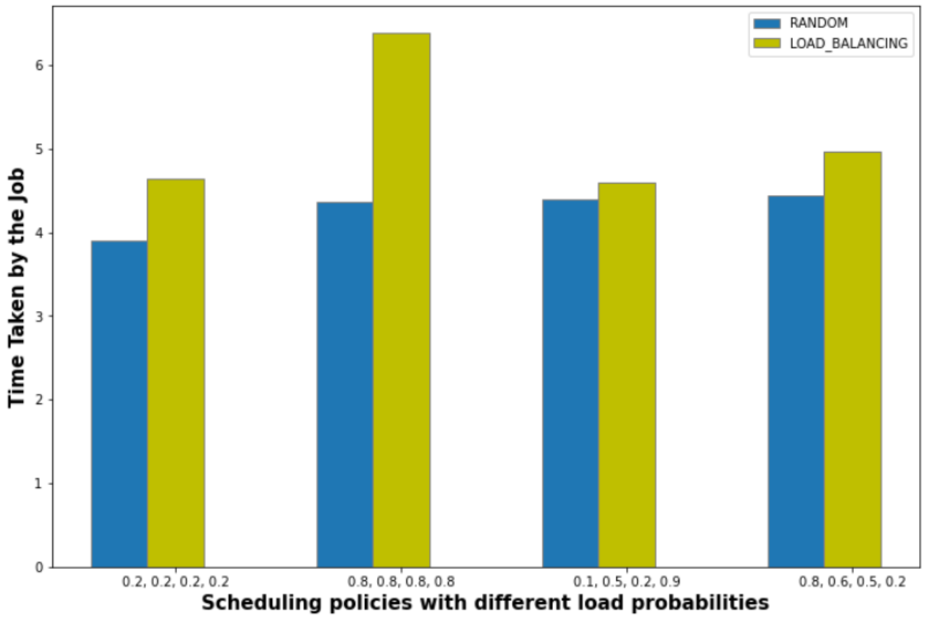
Various datasets are used, with varying sizes and different file types. And for the analysis, we made sure to consider all the edge cases like, all nodes having high load probabilities and low load probabilities, empty input directory, different file types other than images in the input data.

Below are the graphs plotted between different load probabilities and the respective time taken to complete the job for each of the scheduling policies (random and load balancing):

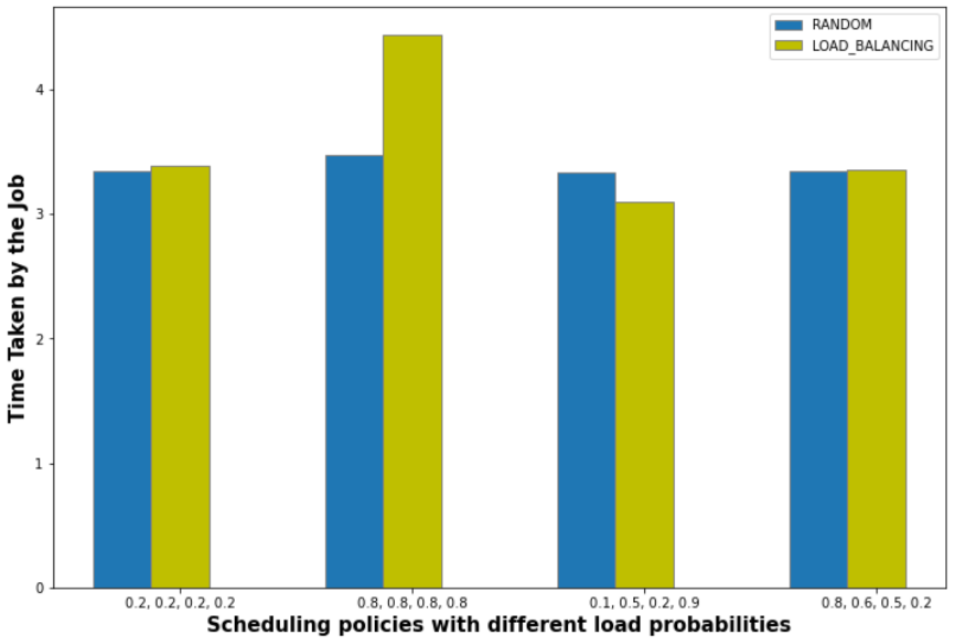
1. Size of the dataset: 103
   Different load probabilities considered : (0.2 0.2 0.2 0.2), (0.8 0.8 0.8 0.8), (0.1, 0.5, 0.2, 0.9), (0.8, 0.6, 0.5, 0.2)



2. Size of the dataset: 50
   Different load probabilities considered : (0.2 0.2 0.2 0.2), (0.8 0.8 0.8 0.8), (0.1, 0.5, 0.2, 0.9), (0.8, 0.6, 0.5, 0.2)



3. Size of the dataset: 15
   Different load probabilities considered : (0.2 0.2 0.2 0.2), (0.8 0.8 0.8 0.8), (0.1, 0.5, 0.2, 0.9), (0.8, 0.6, 0.5, 0.2)



Insights from the above graphs:

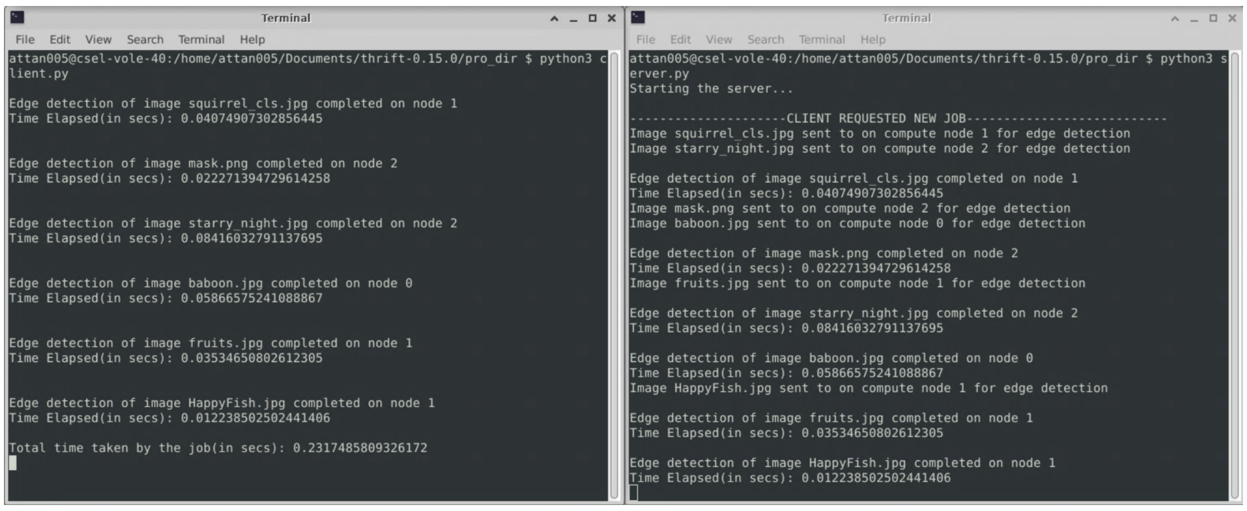
- So from the above graphs, we can see that Load Balancing takes more time compared to Random scheduling policy since in this scheduling policy few tasks might be rejected by the compute nodes and the server has to reassign them to a compute node.
- We can also see that when load probabilities are high for all the nodes, the time taken to complete the task is very high compared to the other, because if the load probabilities are high, then the chance that the task might get rejected is high, and also the probability of delay injection in the task is high which leads to high job completion time.
- If the load probabilities are low, then the job takes comparatively less time irrespective of the scheduling policy.

Negative test cases considered for performance analysis:

1. If the input directory is empty or not present, then the time taken to complete the job is 0.0014 sec, and the output directory will be empty.
2. If the input samples have a different file type other than an image, then the compute node will not perform the image processing task on the file.

SAMPLE RESULTS:
Scheduling policy = Load Balancing
Load probabilities given = (0.1, 0.5, 0.2, 0.9)
Size of dataset = 6

Screenshot of working client and server:

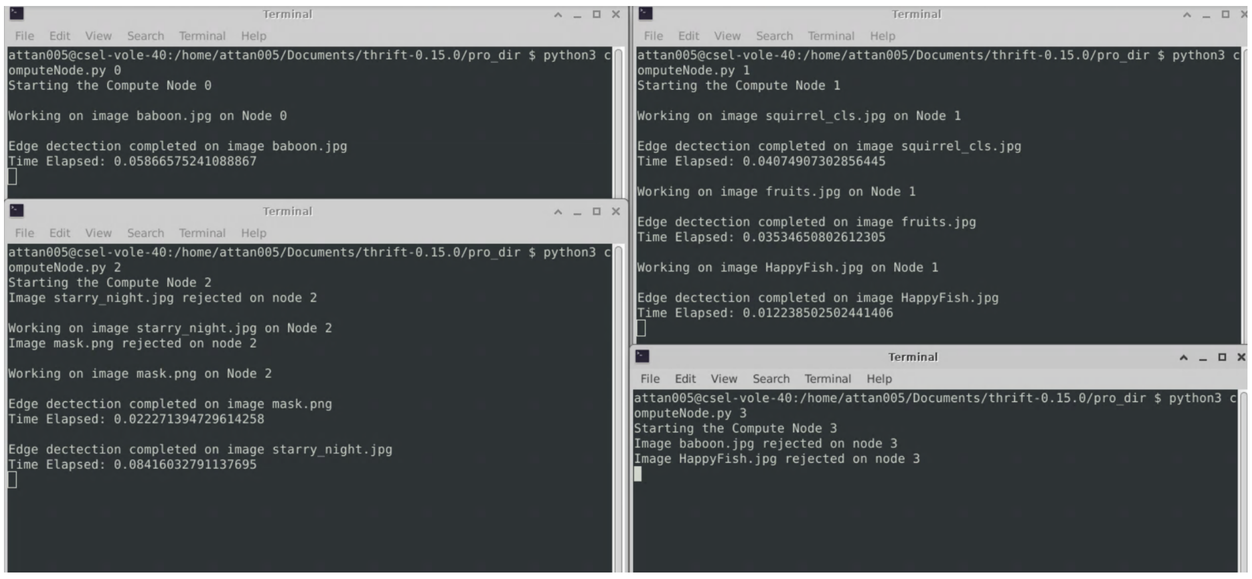


```
attan005@csel-vole-40:/home/attan005/Documents/thrift-0.15.0/pro_dir $ python3 c
lient.py

Edge detection of image squirrel_cls.jpg completed on node 1
Time Elapsed(in secs): 0.04074907302856445


Edge detection of image mask.png completed on node 2
Time Elapsed(in secs): 0.022271394729614258


Edge detection of image starry_night.jpg completed on node 2
Time Elapsed(in secs): 0.08416032791137695


Edge detection of image baboon.jpg completed on node 0
Time Elapsed(in secs): 0.05866575241088867


Edge detection of image fruits.jpg completed on node 1
Time Elapsed(in secs): 0.03534650802612305


Edge detection of image HappyFish.jpg completed on node 1
Time Elapsed(in secs): 0.012238502502441406

Total time taken by the job(in secs): 0.2317485809326172
```

```
attan005@csel-vole-40:/home/attan005/Documents/thrift-0.15.0/pro_dir $ python3 s
erver.py
Starting the server...

--------------------CLIENT REQUESTED NEW JOB--------------------------
Image squirrel_cls.jpg sent to on compute node 1 for edge detection
Image starry_night.jpg sent to on compute node 2 for edge detection

Edge detection of image squirrel_cls.jpg completed on node 1
Time Elapsed(in secs): 0.04074907302856445
Image mask.png sent to on compute node 2 for edge detection
Image baboon.jpg sent to on compute node 0 for edge detection

Edge detection of image mask.png completed on node 2
Time Elapsed(in secs): 0.022271394729614258
Image fruits.jpg sent to on compute node 1 for edge detection

Edge detection of image starry_night.jpg completed on node 2
Time Elapsed(in secs): 0.08416032791137695

Edge detection of image baboon.jpg completed on node 0
Time Elapsed(in secs): 0.05866575241088867
Image HappyFish.jpg sent to on compute node 1 for edge detection

Edge detection of image fruits.jpg completed on node 1
Time Elapsed(in secs): 0.03534650802612305

Edge detection of image HappyFish.jpg completed on node 1
Time Elapsed(in secs): 0.012238502502441406
```

As the communication is synchronous, the client waits for the server to complete its job and return the results.

Screenshot of 4 compute nodes:



```
attan005@csel-vole-40:/home/attan005/Documents/thrift-0.15.0/pro_dir $ python3 c
omputeNode.py 0
Starting the Compute Node 0

Working on image baboon.jpg on Node 0

Edge dectection completed on image baboon.jpg
Time Elapsed: 0.05866575241088867
```

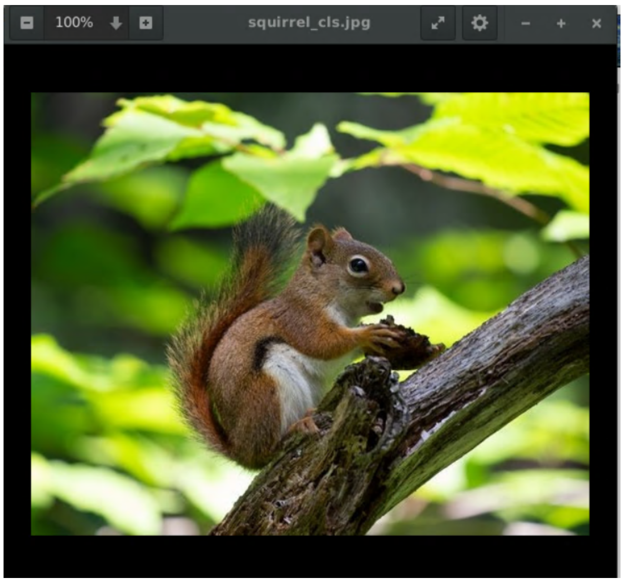
```
attan005@csel-vole-40:/home/attan005/Documents/thrift-0.15.0/pro_dir $ python3 c
omputeNode.py 2
Starting the Compute Node 2
Image starry_night.jpg rejected on node 2

Working on image starry_night.jpg on Node 2
Image mask.png rejected on node 2

Working on image mask.png on Node 2

Edge dectection completed on image mask.png
Time Elapsed: 0.022271394729614258

Edge dectection completed on image starry_night.jpg
Time Elapsed: 0.08416032791137695
```

```
attan005@csel-vole-40:/home/attan005/Documents/thrift-0.15.0/pro_dir $ python3 c
omputeNode.py 1
Starting the Compute Node 1

Working on image squirrel_cls.jpg on Node 1

Edge dectection completed on image squirrel_cls.jpg
Time Elapsed: 0.04074907302856445

Working on image fruits.jpg on Node 1

Edge dectection completed on image fruits.jpg
Time Elapsed: 0.03534650802612305

Working on image HappyFish.jpg on Node 1

Edge dectection completed on image HappyFish.jpg
Time Elapsed: 0.012238502502441406
```

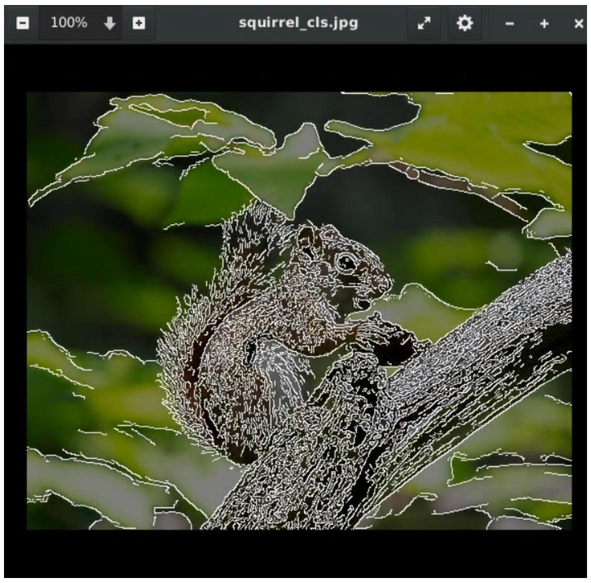
```
attan005@csel-vole-40:/home/attan005/Documents/thrift-0.15.0/pro_dir $ python3 c
omputeNode.py 3
Starting the Compute Node 3
Image baboon.jpg rejected on node 3
Image HappyFish.jpg rejected on node 3
```

From the above screenshot, we can see that, compute node 3 rejects more tasks since we have given a high load probability(i.e 0.9) for that node.

Sample Input Image:



Sample Output Image:


100%
squirrel_cls.jpg