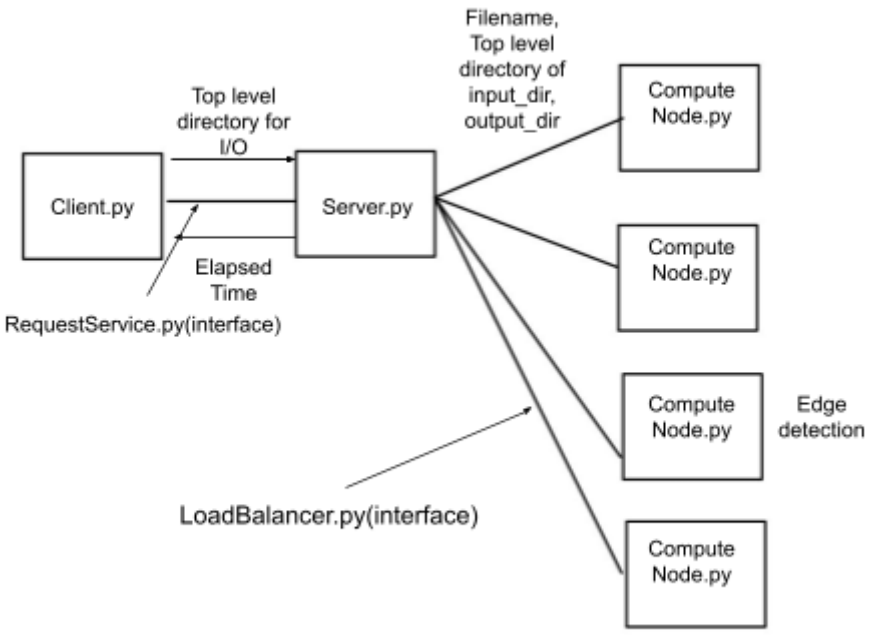
Programming Assignment1 Result Document
X500:song0254, chan2222

**Description of design**

- Client
  - Reads config file to get the directory for I/O
  - Pass the path of the directory to the server, which contains two sub-directories, input_dir and output_dir
  - Submit the job to server by RPC
- Server
  - Iterate through every picture in input_dir, and assign them as jobs to compute nodes parallelly by creating multiple threads according to the scheduling policy
  - Pass the path of the pictures to the compute nodes by RPC
  - Calculate the elapsed time
- Compute Node
  - Maintain its own load-probability according to the configuration file
  - Accomplish the edge detection job by operating Canny Edge Detection, where the method is provided by OpenCV
  - It is implemented as a multi-threaded machine, which means it is able to accept multiple computation tasks from the server



**Usages**

`$ python3 client.py`

- In order to run client.py, you have to change the path of thrift on the top of the file

```
sys.path.insert(0,glob.glob('../lib/thrift-0.15.0/lib/py/build/lib*')[0])
```

`$ python3 server.py`

- In order to run server.py, you have to change the path of thrift on the top of the file

```
sys.path.insert(0,glob.glob('../lib/thrift-0.15.0/lib/py/build/lib*')[0])
```

- Run server to send request to computeNode and return total elapsed time

`$ python3 computeNode.py nodeNumber`

- In order to run computeNode.py, you have to change the path of thrift and opencv on the top of the file

```
sys.path.insert(0, glob.glob('../lib/thrift-0.15.0/lib/py/build/lib*')[0])

sys.path.insert(1, glob.glob('../lib/opencv/build/lib/python3')[0])
```

- Compute Canny edge detection requested by server
- nodeNumber argument specify the node number to assign

**config**

- configuration file which contains host name, port number, scheduling policy, and load probability for each node
- If you run in local, you need to specify the different port inside the config file
- E.g.
    - policy:load_balancing
    - probability:0.8,0.8,0.8,0.8
    - server:csel-kh1250-05.cselabs.umn.edu,9192
    - node_0:csel-kh1250-05.cselabs.umn.edu,9193
    - node_1:csel-kh1250-05.cselabs.umn.edu,9194
    - node_2:csel-kh1250-05.cselabs.umn.edu,9195
    - node_3:csel-kh1250-05.cselabs.umn.edu,9196
    - client:csel-kh1250-05.cselabs.umn.edu,9191

**Assumptions on client**

- Client should be able to provide(create directory) for desired input_dir and output_dir

**Testing**

- Test Script
    - Run `python3 test.py`

    - test.py will read config files from “test” directory to set up different scheduling policies and load probability.
    - Before you run the test.py, please change the proj_path in config file to the proper path
- Test Cases
    - Testings are done in sequential manner, which checks high load probability to low to check performance.
    - Test also includes checking empty input and empty output
    - Test Case1:
        - Policy: Load Balancing
        - Load Probability: 0.8, 0.8, 0.8, 0.8
    - Test Case2:
        - Compare the result from Test Case 2
        - Policy: Load Balancing
        - Load Probability: 0.1, 0.1, 0.1, 0.1
    - Test Case3:
        - Compare the result from Test Case 1, which has various probability and should be faster
        - Policy: Load Balancing
        - Load Probability: 0.8, 0.5, 0.3, 0.1
    - Test Case4:
        - Compare the result from Test Case 1, which no rejection and should be faster
        - Policy: Random
        - Load Probability: 0.8, 0.8, 0.8, 0.8
    - Test Case Empty Input Output:
        - Should get empty output_dir
        - Make sure your input_dir and output_dir are backed up. It will erase all contents on input_dir and output_dir
- Autograding:
    - Before you run autograding, you need to match hostname in machine.txt and in the config file for each node and server.
    - E.g.
        - config: node_0:csel-kh1250-05.cselabs.umn.edu,9193
        - Machine.txt: node_0 csel-kh1250-05.cselabs.umn.edu
    - If you’re running in same machine, you need to configure different port number in the config file

**Results**

- Case1: Random, Load Balancing, 0.8, 0.8, 0.8, 0.8

- Case2: Random, Load Balancing, 0.8, 0.5, 0.3, 0.1
- Case3: Random, Load Balancing, 0.5, 0.5, 0.5, 0.5
- Case4: Random, Load Balancing, 0.1, 0.1, 0.1, 0.1

**Analysis**

- For random policy, the algorithm was able to achieve consistent performance, which converged to the sleeping time of the program. If the load probability is lower(which has less chance to wait), random policy could achieve around 0.05 seconds. Also, as shown in the graph, random policy could achieve faster performance than load balancing policy.
- For load balancing policy, as the load probability got lower, it was faster to process all the images. This result prevailed since the load probability becomes lower, it has less chance the tasks are rejected.