CS 423 MP4 Group 9

Laurynas Tamulevicius (tamulev2)
Pranava Aditya (paditya2)
Vassil Mladenov (mladeno2)
David Lipowicz (lipowic2)

Design and Implementation:

For this MP, we tried to closely follow the instructions and steps given in the MP document. In order to implement the dynamic load balancer, we used the Python language and followed the steps described below.

1. We started off by creating a job queue which we implemented as an array which stores the jobs currently pending and it then also passes them to the worker thread.

2. This is what we then implemented next. The worker thread processes each job and stores the result once done. We also had to make sure that the thread reacted to throttling. We do this in the hardware monitor and so decided to work on that next.

3. The hardware monitor allows the user to set the throttling value and we use sleeping to wait the relevant amount of time until the worker thread should be woken up/put to sleep.

4. Our transfer manager was implemented next. We use it to transfer jobs between nodes. This is done by opening sockets on both nodes and transferring the data between them. Once it receives jobs, it places them in the job queue.

5. Next we worked on the state manager. This is the part of the system that sends the system state to the remote node. We chose an Information Policy of 30 seconds.

6. Finally we worked on the adaptor. The adaptor decided whether a load balance transfer should happen. We do this by using the queue length and throttling value.

Analysis and Experimentation:

When working out the best way to implement this project, there were a few things we had to experiment with. The main one was the node balancing logic. We assumed that **the big node was 4 times faster than the little node.**
We then did the following calculations:

Big node: n jobs, x load_factor
Little node: m jobs, y load_factor

k: big_node / little_node speed
In this case we guessed k = 4

We then used the following equation to decide the number of jobs that need to be transferred:

(n+s) / (k * x) = (m-s) / y

We then solve this equation for **s**, which is the number of jobs that need to be transferred to the other node.
Note: It's going to be an identical calculation for both nodes, except with the opposite sign.

For example:

if s = -150, I should receive 150 jobs
If s = 14, I should send 14 jobs

The next design decision that we had to make was the choosing which transfer policy to use and implement. We used the **symmetric initiated transfer policy** for this MP. In order for this policy to work, both the nodes perform the calculation explained above and the one with the positive number sends the jobs. The data that we used to analyze the performance of this policy is as follows:

Total number of jobs: 512
Size of one job : 8KB
Average Job finish time: 1.58 seconds
Average number of jobs transferred: 36
Average amount of data transferred: 36*8KB = 288KB

After implementing and analyzing this policy, we saw that the results were good and therefore decided to stick with the symmetric initiated transfer policy for this MP.