1 Overview

This document is a record of data structure design of project 3. There are three type of tiers: Master, Front, and Middle. We start with 1 master tier, 1 front tier, and 1 middle tier, and we perform scale in and scale out on middle tiers based on the average length of request queue and the interarrival rate of clients.

2 Roles of server instances

We bind the first VM as master server, and build 1 front tier server and 1 middle tier server.

Master is responsible for managing the request queue. It communicates with each front end server and store the request in its master request queue, and then communicates with middle tier server to give out requests. When initializing the master instance, we also start a daemon thread that works behind the scene and collect the average queue length as a reference for scaling out, which will be detailed below.

Front is responsible for receiving requests from ServerLib, and pass to master.

Middle is responsible for receiving requests from master and process them. Middle also collects the interarrival rate between requests and decide to scale in, which will be detailed below.

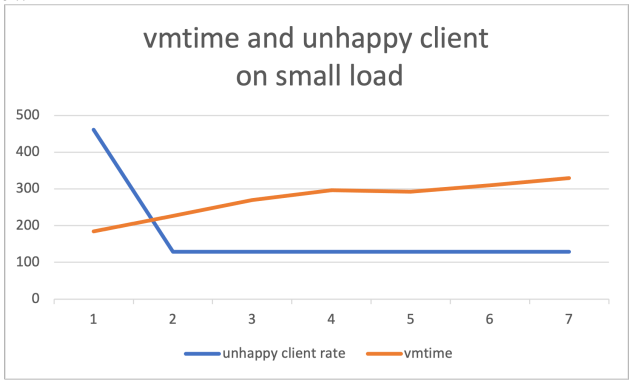
3 Scale out

We start the daemon thread as we start the master server, and keep collecting queue length. Once we collect 10 queue length, we compute the average of them and use intervals to decide how many middle tier servers we need to scale out.

We divide four intervals based on small load, large load, stepping up, and stepping down. When dealing with small load, average of queue mostly falls in 2 to 4.5; when dealing with large load, average of queue falls larger than 7. We further divide two intervals, 4.5 to 6.5, and 6.5 to 7, for better stepping up and down behavior.

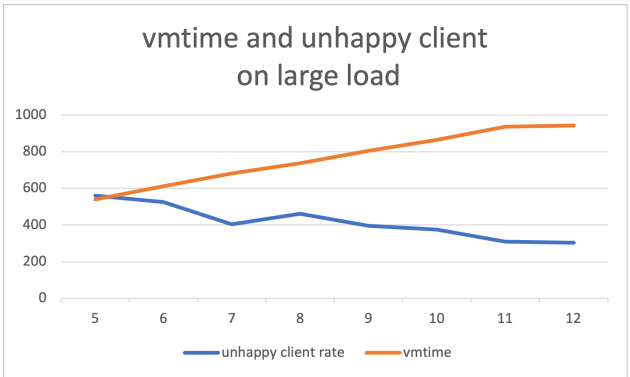
We perform benchmarking with different load to decide the amount of scale out we need. We calculate the unhappy client rate by 1000*fraction of (dropped+timeout+failed)/total amound, and calculate VMtime by total vm time / 1000.

When dealing with small load, the change of unhappy client rate and change of VMtime appears as below:



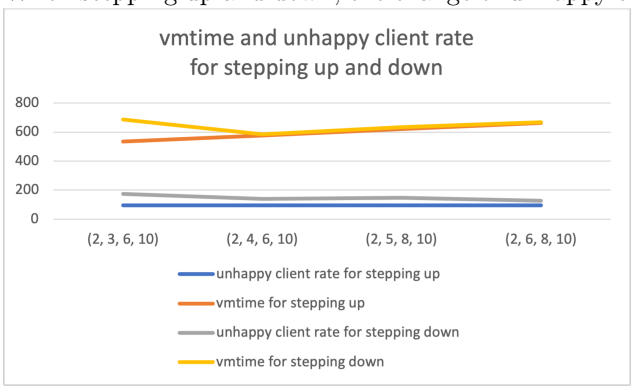
We can see that the unhappy client rate drop drastically at vm = 2 and then remain constant, and vm time increase slowly as vm number increase. Therefore, we choose vm = 2 as the number of middle tiers we scale out when dealing with small loads (i.e., when average falls in 2 to 4.5).

When dealing with large load, the change of unhappy client rate and change of VMtime appears as below:



The unhappy client rate drops fast as vm increase from 5 to 7, slightly increase, and keeps dropping after vm = 8. The vmtime keeps increasing from vm = 5 to 11 and remain constant after vm = 11. We scale out 10 vms when dealing with large loads (i.e., when average is larger than 7).

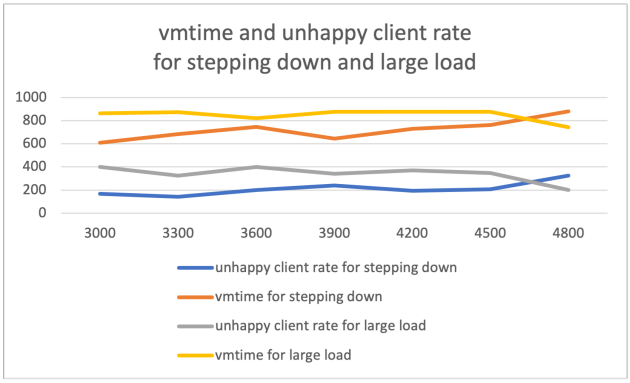
When stepping up and down, the change of unhappy client rate and change of VMtime appears as below:



The unhappy client rate for both stepping up and down seems to be approximately constant, the vmtime for stepping up seems to be always increasing, and the vmtime for stepping down reaches min at (2, 4, 6, 10) and starts increasing at the same slope with the vmtime for stepping up. Therefore we choose 4 as the number of vms to scale up for average falling in 4.5 to 6.5 and 6 as the number of vms to scale up for average falling in 6.5 to 7.

4 Scale in

We use the difference in arrival time between each two clients as the standard to scale in. We conduct a benchmark on the difference of time between each two clients when stepping down and large loads.



Since the unhappy client rate reaches min at diff = 3300ms, we choose 3300ms as the boardline for scale in.