Name: Ming Yao
Andrew ID: mingyao

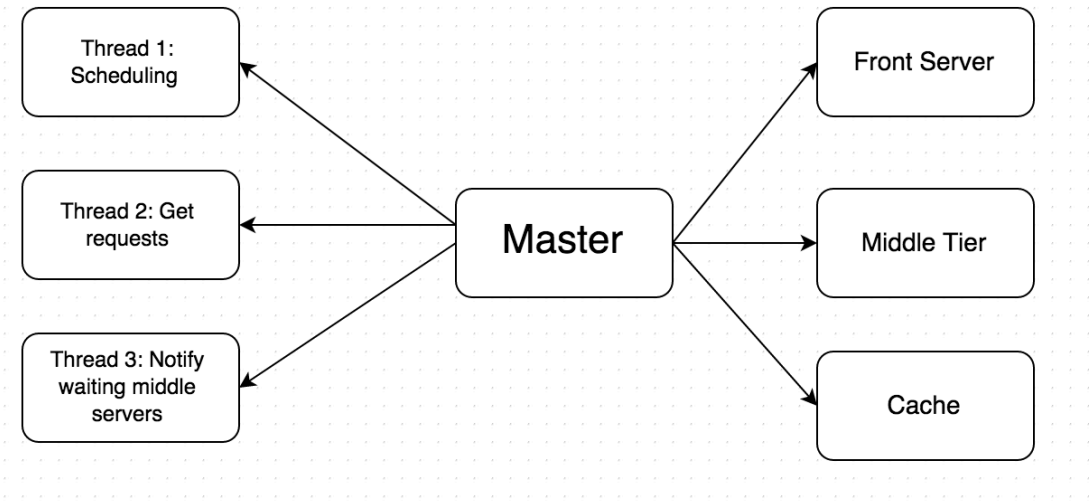
15-640 Distributed Systems Project 3 Report

1. Architecture

The whole architecture of the program is like the following diagram. In master node, we have three threads. The first thread is for scheduling, the details of scheduling is in the next section. The second thread is getting request and put them into a global request queue, just like other front servers. I implemented a **global request queue** on the master node, when other front servers get requests, they push the requests into the global queue by RMI, while master node can directly put requests in this queue. Then when a middle tier server is ready to process a request, it will get a request from the master by pull a request from the global queue.

The third thread is for notifying waiting middle tier servers. When middle tier server wants to get request from master but the queue is empty, it will get blocked. The blocked servers is stored in a **ReadyServer list**. Then when the queue is not empty, this thread will notify the first server in the ReadyServer list. So the middle tier server can continue to process the request.

By detecting the queue length and middle server average waiting time, the master node can decide whether to scale out or scale in an front end server/middle tier server. It will also bind a cache to accelerate processing the get requests.



2. Scheduling

We scale out front server and middle tier server separately. For scaling out front end server, the criterion is: we first gather all the front servers' queue

length (by calling SL.queuelength, it is actually the queue length of the cloud, not the global queue on the master node). When the total queue length is larger than half of the number of front servers (include master), we add a front server.

For scaling out a middle tier server: when the global queue length is larger than the number of middle tier servers, it's an indication. We look at the queue length every 100ms, it this happens 5 consecutive times, we add middle tier servers. In order to scale quickly, we add a scale factor. The number of middle servers we will add is factor*(average queue length of 5 times/number of existing middle servers). The first time it should scale more, so we give a higher factor for the first time.

For scaling in, I use a different method. Every time when a middle tier server is ready and want to get a request from master, it will cost some time. When the queue is empty, it will get blocked so the **waiting time** would be longer. The master node check every waiting time and record them in a sliding window. To be specific, we sum 5 consecutive waiting time, check the sum every 100 ms for 5 times, if the average sum is larger than a thredshold, then we shutdown a middle server. Actually, I scale in the front server with the middle server together. But it has a minimum number of front servers and midde servers (i.e., 2 middle tier servers and 2 front servers(including master)).