1. Master Instance

Master instance is the leader of this cluster. Master is responsible of

1.1 Scaling up the middle tier if necessary.

Master is the manager of the cluster. In my implementation, there is a daemon thread collecting the queue length every second. Master will use those data to scale up the middle tier based on heuristic. Those heuristic can be get from a number of off-line experiments.

1.2 Managing the request queue.

Master is responsible for managing the request queue. So every front instance has to communicate with master to add request to the queue. The other design I considered is to scatter the requests to all the middle instances. The advantage is that it releases some work from master. The worse thing is that we have to communicate with all the middle tiers to get the current queue length in order to decide whether to scale up or down, which is very tricky. I choose to put all the requests on master because it is much easier to implement and debug. Most of distributed system usually avoids scattering the information unless necessary.

1.3 Managing the Cache

Besides the role of coordinator, master is also responsible for the cache tier. In my implementation, I cache the <key, value> pair in the cache instance for fast retrieval.

2. Front Instance

Front tier instances are mainly responsible for getting the requests from the Server Lib and resend them to the master.

3. Middle Instance

Front tier instances are mainly responsible for getting the requests from master and process them. Besides, middle tier instances will measure the time interval between two processes. If the interval is smaller than the threshold (1500ms), then it will shut down itself and tell master that it is gone.