Servers

There are three roles of servers, master, front and middle servers.
All the server share the same address with master servers, front and middle servers are VMs.
Master controls the number of front servers and middle servers.
Master, it first initialize lists of front servers, middle servers and request queue.
Initially is a front server, we register front end because we want to know the come in rate of the first two request, it then starts a front server and a middle server, master drops all the requests in the SL before middle server and front servers start, because we cannot process these requests without starting the front and middle servers.
It then test the interval come in rate of the first two coming requests. The init number of middle servers and front servers benchmarked on the interval.
And then it unregisters its front server role, focus on controlling the number of front servers and middle servers.
In the big while loop in the master, it scale out or scale in the number of servers based on the interval rate and the request queue.
front servers simply add request to the master request queue using RMI.
middle servers gets the next request from master request queue and process the request using RMI.

Scale In/Scale Out

Compare the length of requestQueue with number of middle server, if it is longer than middler servers. We scale out, that is add middler servers, since front server only get request from the master server, it should not take too long, so we do not care about adding front servers.
If the request queue is still bigger than the middle server, we cannot keep up the speed, it means we have to drop some requests in request queue since we are not able to finish it, we want to avoid purchased after timeout.
Then we test the come in interval of the next two requests.
Based on the interval rate over some period of time and some number of requests, if it is faster than before, we scale out, that is add more servers.
If the rate is smaller than before, we scale in, that is delete some servers.

Cache

Cache is implemented based on the interface of Cloud.DatabaseOps
We have a conconcurrent cacheMap storing all the keys and values
get is implemented by first try to get the value from the cacheMap, if it is not in the map, we get it from the database and add it to the map
set is implemented by first setting the value in the database, and then update the values in the cachemap
transaction is implemented by returning transaction of the database