Project 3 Scalable Service

Name: Xiangyu Sun
Andrew ID: xiangyus

1.General Design

There are two tiers in this system, front tier and middle tier. Front tier is responsible for retrieving requests from Cloud. Middle tier is responsible for processing requests.

This system is able to scale out and in to satisfy the incoming requests in less VM times. There exists a master VM acting three roles which is front tier server, scalable management server and cache database server. Also the master maintains a request queue. All the other front tier servers retrieve the request, put a time stamp with it and store it in the request queue. All the middle tier servers retrieve requests from master and check how many seconds are left to decide to drop it or process it.

2.Scalable Strategy

Master open a thread to do the scalable management. It measures the rate of the incoming requests and according to this to calculate the request per second. Using this RPS metrics we can decide whether the number of our middle tier machines is enough or not. And scaling in shouldn't happen in the first 30 seconds and less than 7 seconds after VM is shut down. Scaling in signals must reach 20 times to actually scale in.

3.Cache Implementation

I put a hashmap on Server class to act as cache and use Server class to extend Cloud.DatabaseOps. The server will also bind the service using RMI. Each mid tier server will need to use RMI to get the instance of Cache and use it as a parameter when calling the method processRequest().

4.Role Coordinate

The master has three queues for roles. One stores all the unassigned roles. One stores thre roles of existing mid tier servers. One is for front tier servers.. When scalable thread detects lack of mid tier, it will first add mid tier role(RoleMidTier) to the queue and start

corresponding number of VMs. When a new VM is started, it will first connect to master to retrieve its role and use the role name and id to bind its service. Then add its role to the corresponding queue in master to act as an evidence to let master know it has started. And master can use this evidence to let one server shut down itself by RMI.

5.Other details

I use concurrenthashmap as cache to ensure thread safety. I also use LinkedBlockingDeque to store requests. So when queue is empty, the mid tier will wait until it is not. And this will reduce useless RMI calls.