Haoyang Yuan
haoyangy
15-640 / PIT
04/02/2016

Report for Project3

This report will describe the implementation of the scalable cloud servers, including server coordination, scale out scheme, scale in scheme, cache implementation.

1. Server Coordination:

All the servers are coordinated through a master server, and they are also communicating by the RMI manipulation with the master server. The master server works as a front server, and it also contains a concurrentBlockQueue to store all the requests sent from all other front servers. At the same time, all the middle servers will communicate with the master by poll out the request and process them. The master keeps a record of all the servers and counts their status.

2. Server Numbers:

The number of servers each tier to run depends on the client situation. Generally we assume the two front servers are strong enough to handle all requests. And the middle server we assume that each server should handle 5 requests each second.

3. Scale Out Scheme:

Front Server: We will start two front servers initially, including the master server, and whenever each loop in mater detects that the requests queue length of load balancer is 5 times more than the number of front servers, we will scale out one more front server.

Middle Server: We will start one middle server initially. During the booting time, when there is no middle server starts working yet, we will count the coming requests, and scale out a middle server when the 6 requests come. And during the working period, when there is at least one server working, we will count the consecutive requests drop. (assumption: the middle server will drop request when master queue length is larger than the number of middle servers) Whenever there are two consecutive drop detected by the server, it will scale out a new middle server.

4. Scale In Scheme:

Front Server: Due to the larger processing capacity of the front server, we will not scale in front server, and we assume two front servers will be enough to work properly. So we only scale out for it.

Middle Server: We will use LinkedBlockQueue which will wait for the poll result to be available for a given time until timeout and return a null. So we will count the null requests, whenever there are 3 consecutive null requests, which means the middle server is actually not receiving valid requests for 1800ms, the server will be shutdown.

4. Database Cache:
The cache is a simple implementation, it will use a map to record all the get operation results from the database, and store the them locally. So all the future get requests will handled by cache to return the data from map. And for the set and transaction operation, the request will be delivered to the database. An issue here is that we must assume the item name and price is not changing. And even the quantity is not accurate anymore in the cache after someone buys it, the return value of transaction will be determined by the database, so this will work fine in the real purchase behavioral.