To organize my VMs and requests through the front end and application tier, I decided to centralize and manage everything on a master VM. The master VM has 2 ArrayLists that are able to manage the 2 VM tiers independently. These ArrayLists help keep track of VM IDs for each tier that are currently running. Additionally, I have a hashmap to keep track of the roles of each VM that is created. When a new VM is created, it is able to call a RMI method to check its role to properly set up for frontend or the middle tier.

For VM scaling, I decided to benchmark various values, starting from an arrival rate of 0.5 and incrementing up to an arrival rate of 8. I tested various combinations of frontend and middle tier VMS, compiling results in a spreadsheet to determine the best combination. A portion of the spreadsheet is shown below as an example:

Arrival Rate (per second)	Front VMs	Middle VMs	Purchased	Ok	Timeout
1	1	1	9	6	14
	1	2	14	11	4
	1	3	14	11	4
	2	2	14	11	4
2	1	2	22	21	16
	1	3	25	22	12
	1	4	25	22	12
	2	3	25	22	12
3	1	3	36	26	27
	1	4	42	26	21
	1	5	42	26	21
	2	4	42	26	21
0.5	1	1	10	4	0
0.75	1	1	13	8	1
	1	2	13	8	1
	2	1	13	8	1
4	1	4	36	32	51
	1	5	40	38	41
	2	4	36	30	53
	1	6	43	39	36
	2	6	46	41	32
	2	7	44	49	36
	2	5	46	42	38
5	2	5	46	28	78
	2	6	52	38	58
	2	7	57	49	42
	2	8	58	48	41
	2	9	58	50	40
	3	8	57	47	44

After finding the optimal combination of VMs for various arrival loads, I create a 3-part piecewise function to start the optimal number of VMs initially. For dynamic scaling, I compared the difference in queue length for both the front end and middle tier queues over a span of 5 seconds to determine whether or not to scale up/down. If the queues grew relative to the number of VMs in the tier, I decided to scale up an additional VM to be able to get more happy clients. If the length of the queues shrunk compared to the length 5 seconds ago, I would scale down by one VM in the tier.

To deal with hysteresis, I decided to only make changes every 5 seconds, which is how long it takes for VMs to boot up. By doing this, I avoid overreacting to large changes in arrival rate over a shorter span like a second or two.

Over the course of this project, I learned the benefits of scaling out to multiple tiers as well as the benefits to scaling out each tier. By using a multi-tiered approach to the web service, we were able to divide the work between parsing and processing the requests because processing each request is quite slow and costly. It allows for our webservice to have each tier be more task specific, making the overall

system more efficient since there could be VMs to specifically accept requests while other VMs could process those requests. In terms of scaling out each tier, it was important to know how many VMs were needed through benchmarking so that the most amount of requests could be processed before timing out. By dynamically scaling each tier in and out based on the arrival rate, my system could efficiently handle most of the clients while also using the minimum amount of VMs to do so.