**Distributed Task Execution Framework**

Done by

Chaithanya Kumar Anichetty Murali (A20327683)

Shivakumar Vinayagam(A20341139)

Vinodh Kandaswami(A20334757)

**Program Design:**

The goal of this assignment is to develop a distributed task execution framework which is highly scalable, reliable and cost efficient.

The main components of the framework are :

* The Client.
* Front-end Scheduler.
* Dynamic provisioning.
* Local workers.
* Remote workers.
* Animoto Clone.

**The Client:**

The client is a simple command line interface which submits tasks to the front end scheduler. The client connects to the scheduler with the IP address and the port on which the scheduler is running. The tasks are read from a file, these tasks are submitted to the scheduler one by one.

IT get the user task name and the no of tasks that should be put in the file named tasks.txt and then use the file automatically send from the file to scheduler . It also outputs the results from the server in to a file named output.txt.

**Front-End Scheduler:**

The front end scheduler receives tasks from the client and puts in a:

* In-memory queue in case of a local backend worker which resides in the same Virtual Machine as the scheduler.
* In a distributed queue(i.e.) SQS (Simple Queuing Service) in case of remote

Worker.

Tasks are encoded and decoded using JSON to provide interoperability before sending the messages (i.e) the tasks from the client. The scheduler also reads the response from the workers and sends to the client to indicate the completion of the task.

**Dynamic Provisioning:**

This component monitors the SQS and launches remote workers (EC2

Instances) based on the following factor:

* If the queue size is increasing launches remote workers.
* Else doesn’t launch any workers if the queue size is stable or decreasing.

**Local Workers:**

Local worker is implemented in JAVA and resides on the same Virtual machine as the scheduler.

* The Local worker takes tasks from the in-memory queue, executes it and sends a response message to the scheduler.
* The local worker is multi-threaded the number to be run is decided by the frontend scheduler.

**Remote Workers:**

Remote worker runs on a different VM from that of the scheduler.

* Remote worker reads messages from the SQS.
* Checks if the message is present in the DynamoDB with a unique taskID.
* If the message is already present then it discards the message to eliminate

Duplicate tasks.

* If the message is not present, stores the message in the DynamoDB and executes it and sends a message to the response SQS.
* Shutdowns itself if it is idle for more than 20 Seconds.

**Animoto Clone:**

The Animoto clone creates a video from a list of image URLs sent from my client.

* Downloads images to the local disk using the URLs sent by the client.
* Uses ffmpeg to convert these images to a video.
* Stores the video on a S3 bucket and generates an URL for the video.
* Sent this URL to the response SQS, the scheduler reads this message and sends this to the client.
* The client uses these URLs to download the video.

**Manual:**

Program requirements:

For starting Client on a separate VM with the client.jar

1. First execute the command for file creation java -jar client.jar pa4.filecreate
2. It will ask for “enter the no of task to be written to file”
3. It will ask for “enter the which task to be written”
4. It creates the file under the name “Tasks.txt”.
5. Run the jar file in the client machine with command java -jar client.jar
6. It will ask for “enter the ip address” of the server.
7. It will ask for “enter the port no” of the server.

For starting the server on a separate VM with the serverclient.jar

1. It will ask for “enter the port no” for the server to listen to.
2. It will ask for “enter the no of local worker (enter 0 for dynamic provisioning in remote)”.
3. Enter the value of the number of local worker or enter 0 for dynamic provisioning.
4. It will ask for “enter the sleep time of remote worker”.
5. This sets the wait time for the remote worker before it shuts down.
6. Then it waits for the client to contact it.
7. And Starts a thread for each client

For benchmarking use the Remotestaticserver.jar for remote static workers.

For benchmarking use the serverclient.jar.jar for local workers.

For benchmarking use the serverclient.jar.jar for dynamic provisioning.

**Performance Evaluation:**

**LOCAL WORKERS:**

**Throughput of the Local Workers:**

|  |  |  |
| --- | --- | --- |
| No of workers | time | Throughput for 100k sleep(0) |
| 1 | 2.378 | 42052.14466 |
| 2 | 2.366 | 42265.42688 |
| 4 | 2.361 | 42354.93435 |
| 8 | 2.358 | 42408.82103 |
| 16 | 2.351 | 42535.09145 |

**Efficiency of the Local Workers:**

|  |  |  |
| --- | --- | --- |
| No of Tasks | Time taken | Efficiency of sleep (1 sec) |
| 80 | 80.023 | 0.999712583 |
| 160 | 80.028 | 0.999650122 |
| 320 | 80.031 | 0.99961265 |
| 640 | 80.0336 | 0.999580176 |
| 1280 | 80.0363 | 0.999546206 |

|  |  |  |
| --- | --- | --- |
| No of Tasks | Time taken | Efficiency of sleep (2 sec) |
| 40 | 80.014 | 0.999825031 |
| 80 | 80.025 | 0.999687598 |
| 160 | 80.028 | 0.999650122 |
| 320 | 80.032 | 0.99960016 |
| 640 | 80.0339 | 0.999576429 |

|  |  |  |
| --- | --- | --- |
| No of Tasks | Time taken | Efficiency of sleep (4 sec) |
| 20 | 80.01017 | 0.999872891 |
| 40 | 80.016 | 0.99980004 |
| 80 | 80.027 | 0.999662614 |
| 160 | 80.029 | 0.999637631 |
| 320 | 80.034 | 0.999575181 |

|  |  |  |
| --- | --- | --- |
| No of Tasks | Time taken | Efficiency of sleep(8 sec) |
| 10 | 80.008 | 0.99990001 |
| 20 | 80.0096 | 0.999880014 |
| 40 | 80.012 | 0.999850022 |
| 80 | 80.026 | 0.999675106 |
| 160 | 80.028 | 0.999650122 |

**REMOTE WORKERS:**

**Throughput of the Remote Workers:**

|  |  |  |
| --- | --- | --- |
| No of workers | time | Throughput for 100K sleep(0) tasks |
| 1 | 844.145 | 118.4630603 |
| 2 | 795.756 | 125.6666616 |
| 4 | 768.446 | 130.1327614 |
| 8 | 763.652 | 130.9496996 |
| 16 | 757.129 | 132.077889 |

**Efficiency of remote workers:**

|  |  |  |
| --- | --- | --- |
| sleep(1) |  |  |
| No of Tasks | Time taken | Efficiency |
| 80 | 94.327 | 0.848113478 |
| 160 | 105.625 | 0.75739645 |
| 320 | 116.812 | 0.684861144 |
| 640 | 124.117 | 0.644553123 |
| 1280 | 130.548 | 0.612801422 |

|  |  |  |
| --- | --- | --- |
| sleep(2) |  |  |
| No of Tasks | Time taken | Efficiency |
| 40 | 91.784 | 0.87161161 |
| 80 | 95.425 | 0.838354729 |
| 160 | 104.895 | 0.762667429 |
| 320 | 114.978 | 0.695785281 |
| 640 | 125.624 | 0.63682099 |

|  |  |  |
| --- | --- | --- |
| sleep(4) |  |  |
| No of Tasks | Time taken | Efficiency |
| 20 | 88.462 | 0.904343108 |
| 40 | 91.544 | 0.873896705 |
| 80 | 94.372 | 0.847709066 |
| 160 | 103.768 | 0.770950582 |
| 320 | 112.547 | 0.710814149 |

|  |  |  |
| --- | --- | --- |
| sleep(8) |  |  |
| No of Tasks | Time taken | Efficiency |
| 10 | 85.977 | 0.930481408 |
| 20 | 88.954 | 0.899341233 |
| 40 | 90.781 | 0.88124167 |
| 80 | 95.248 | 0.839912649 |
| 160 | 102.684 | 0.779089245 |

**ANIMOTO:**

|  |  |
| --- | --- |
| No of Workers | time for 160 jobs |
| 1 | 13419.865 |
| 2 | 6774.619 |
| 4 | 3519.754 |
| 8 | 1677.457 |
| 16 | 882.945 |

**Dynamic Provisioning:**

For my dynamic provisioning logic it took about **3.542** sec to execute the task and show the output on the average of 5 values.

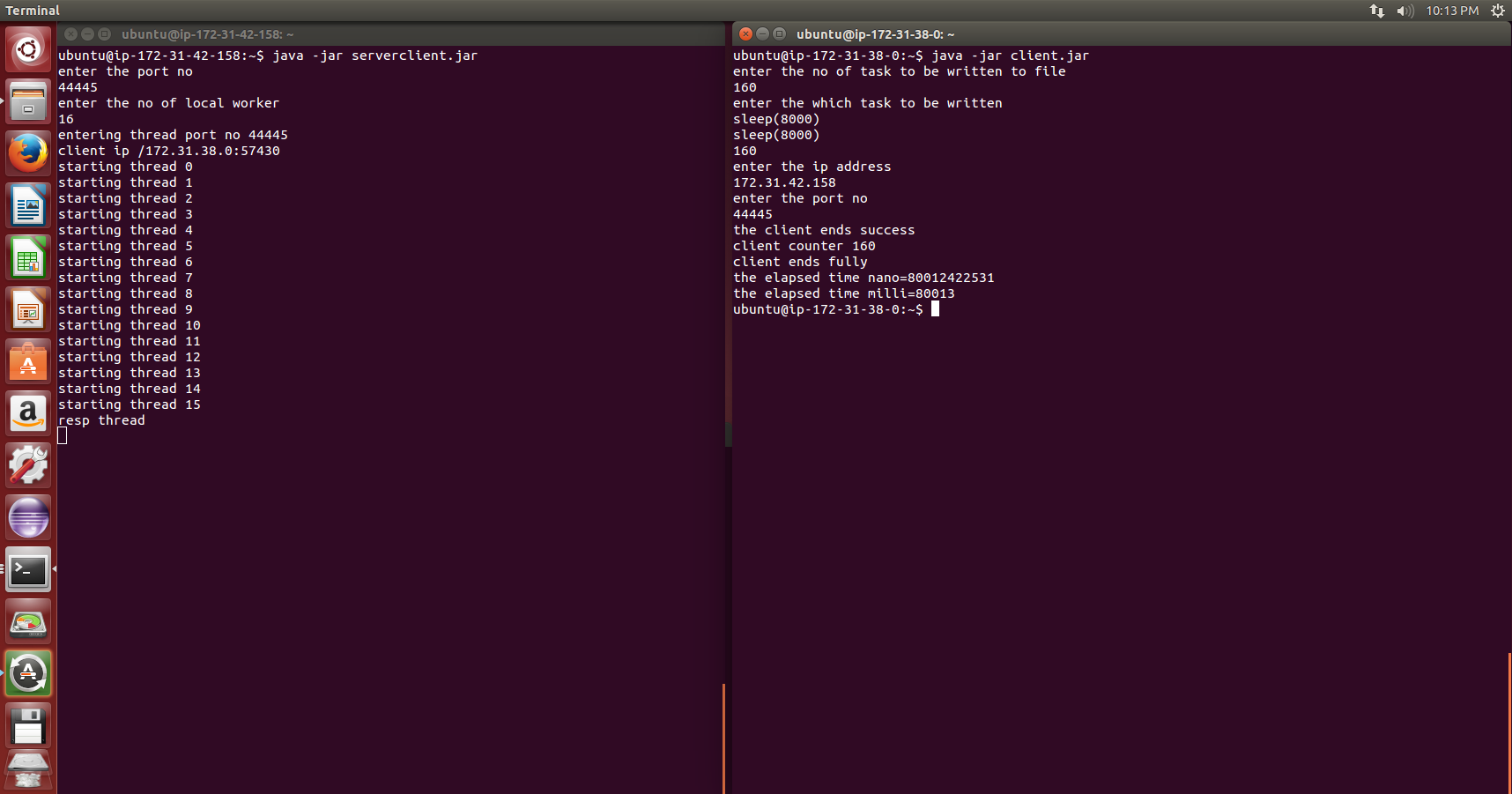
For the Static provisioning it took about **2.351** sec to execute the task and show the output.

Hence my logic is a little bit slower than the static provisioning logic.

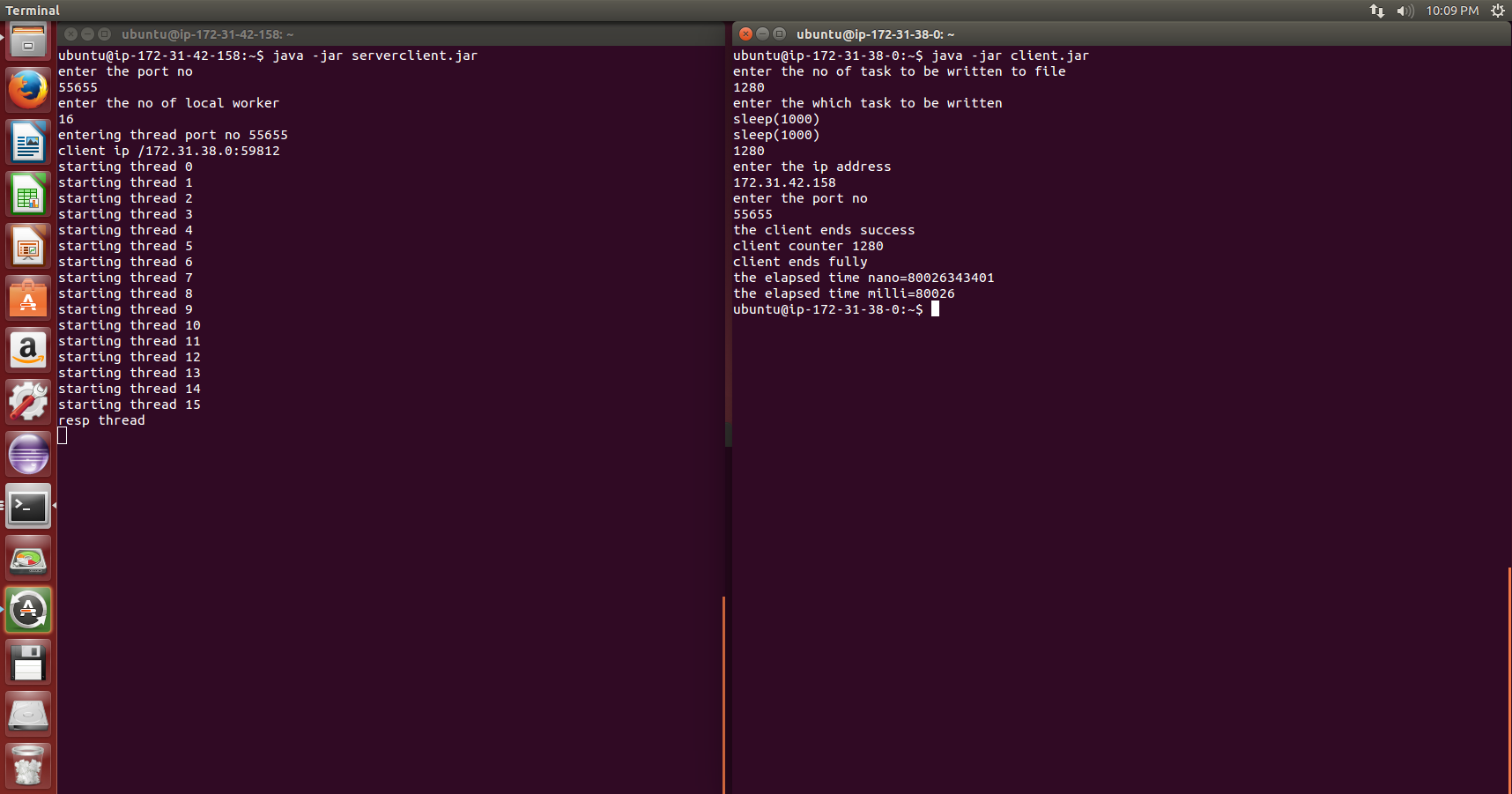
**SCREENSHOTS:**

**Server and Client side shots:**

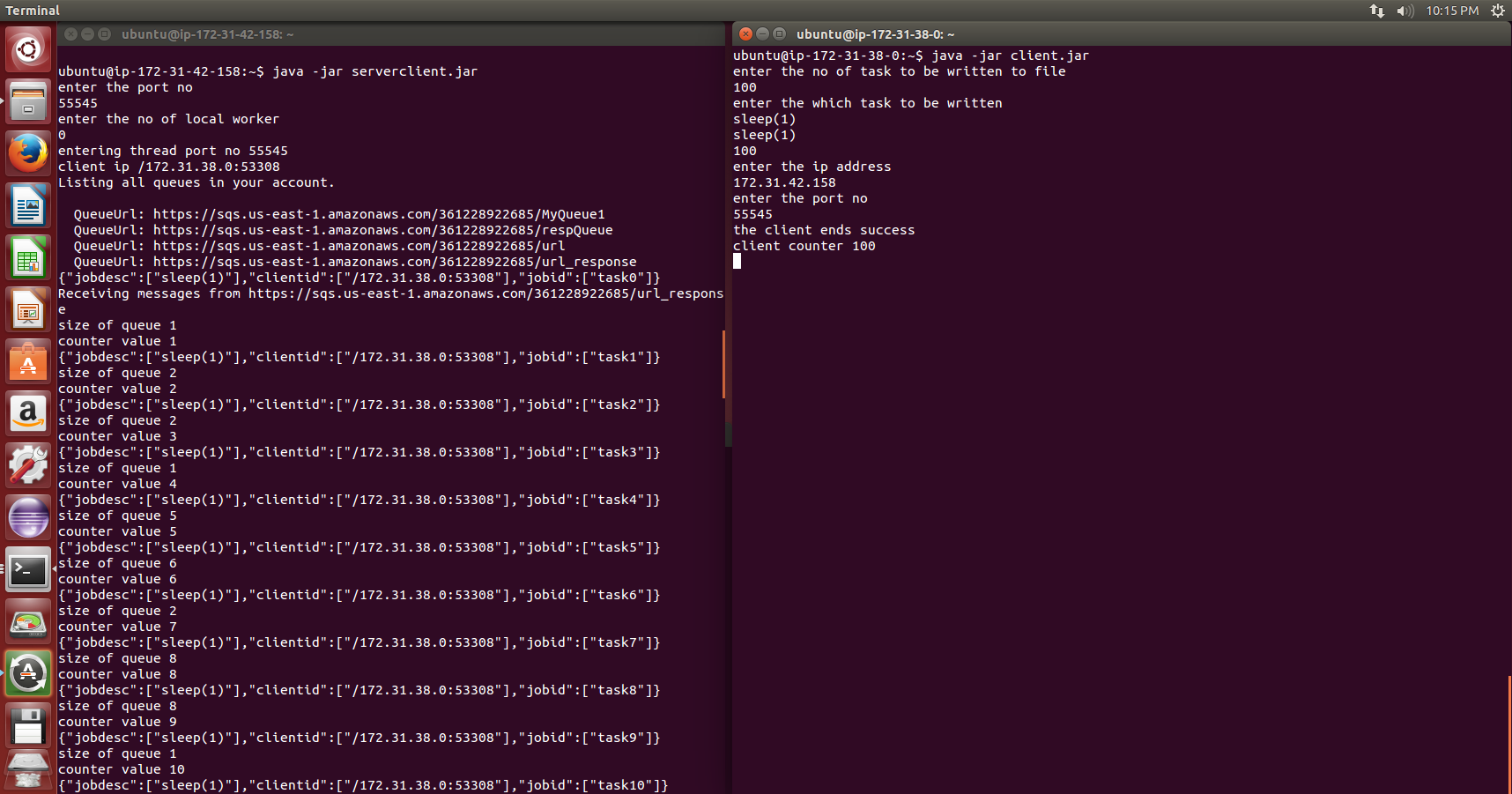
16 threads local :



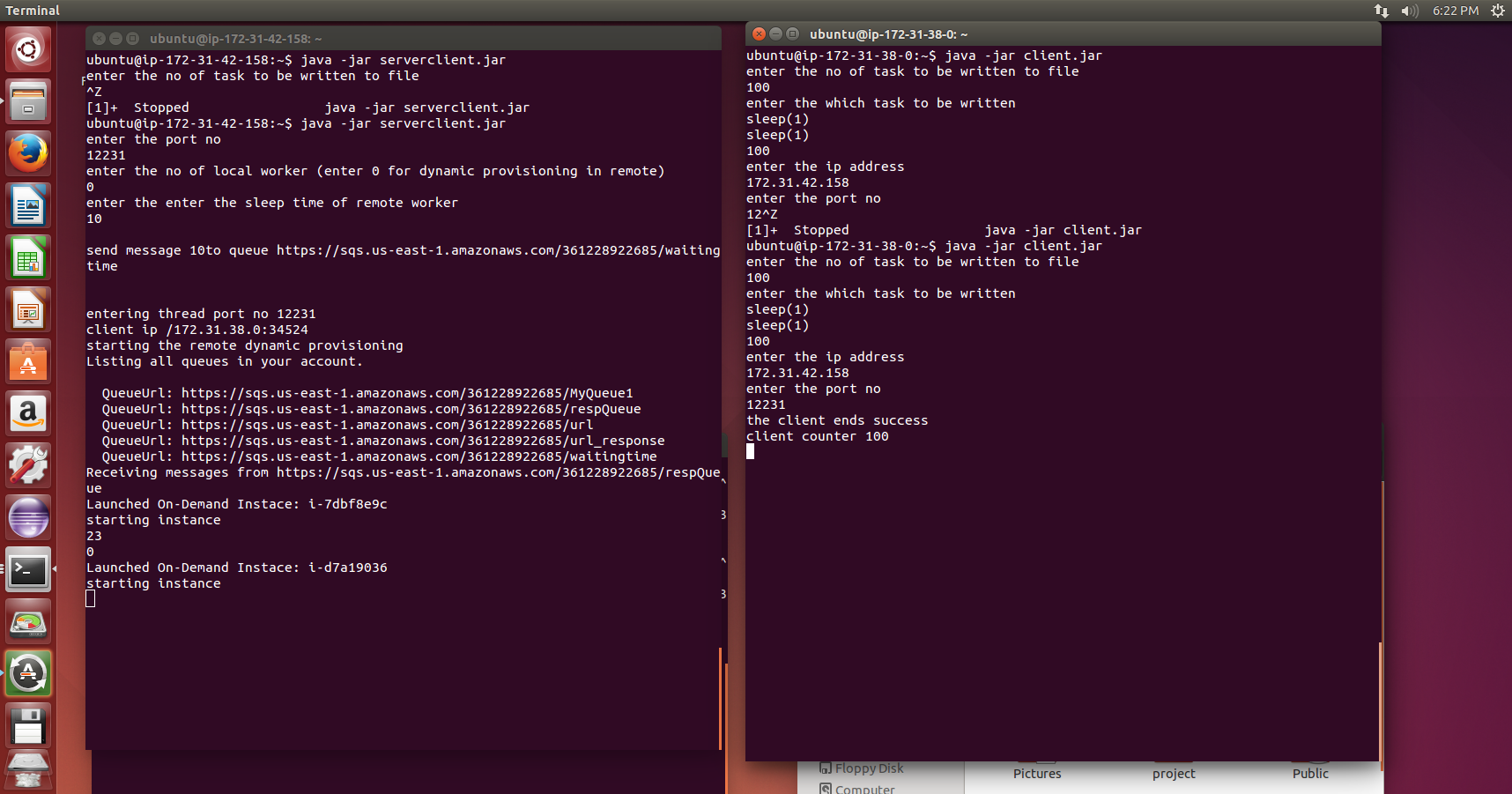
1280 sleep(1) tasks single thread

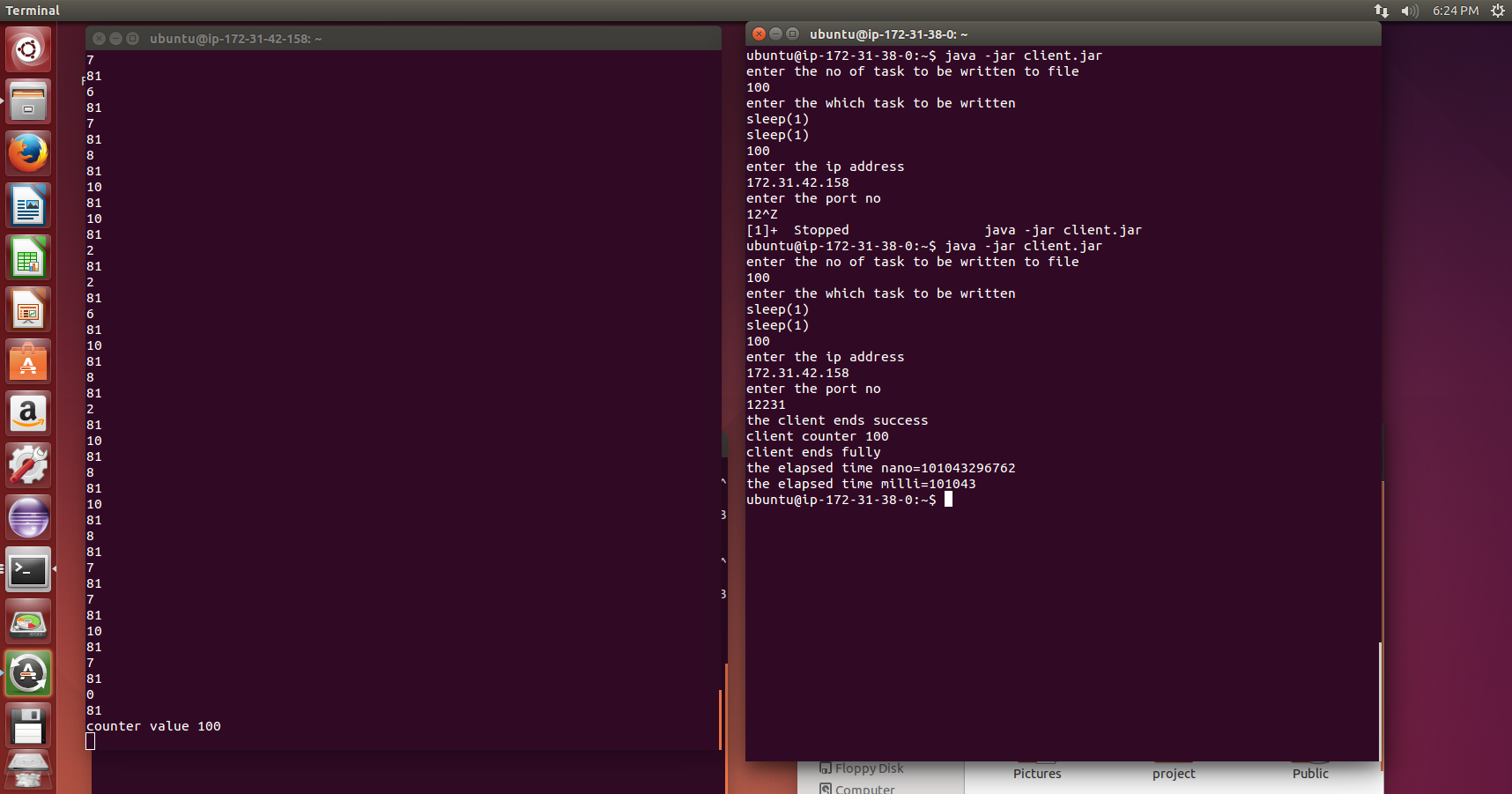


Static Provisioning remote :

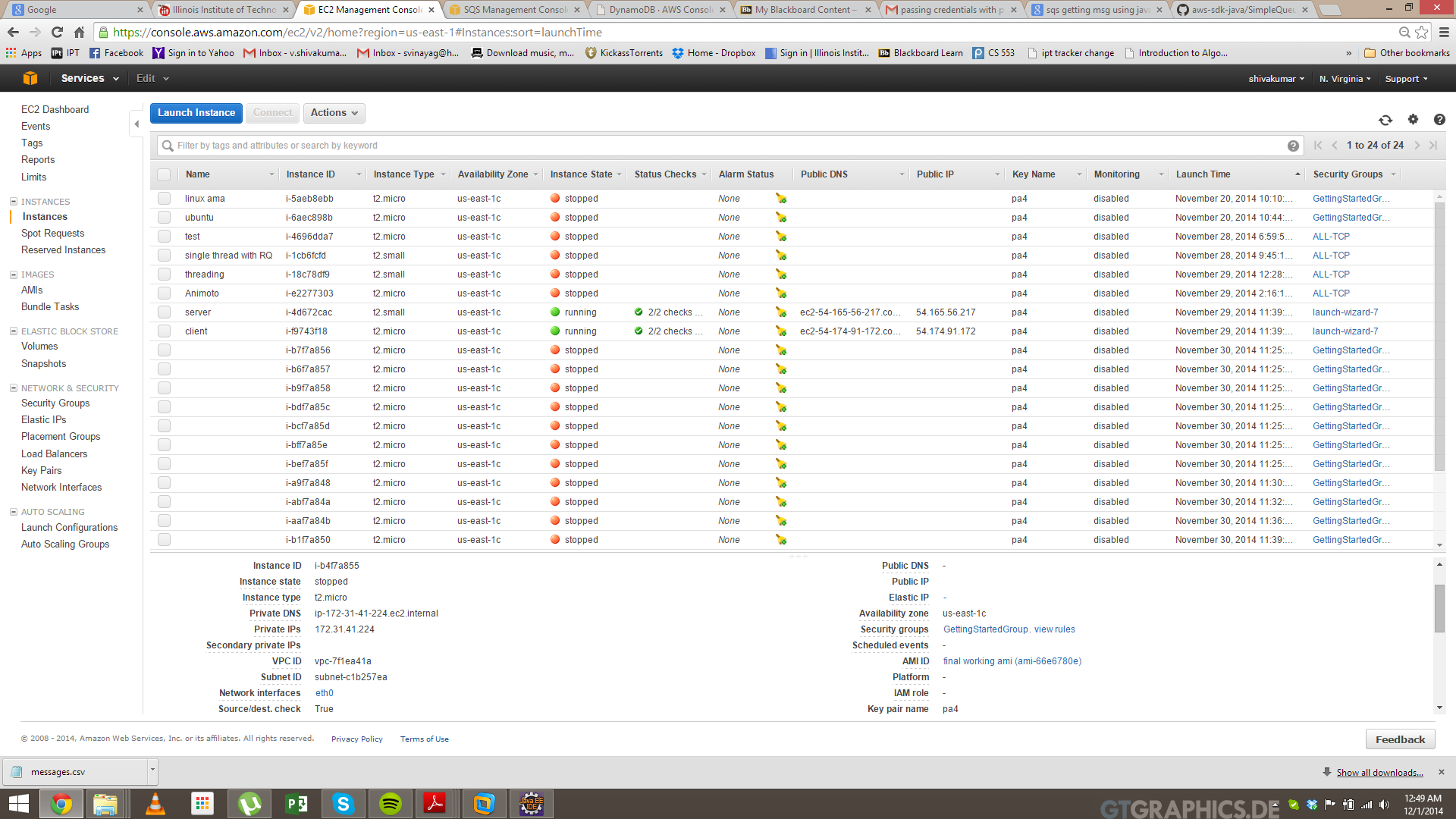


Dynamic Provisioning Remote:

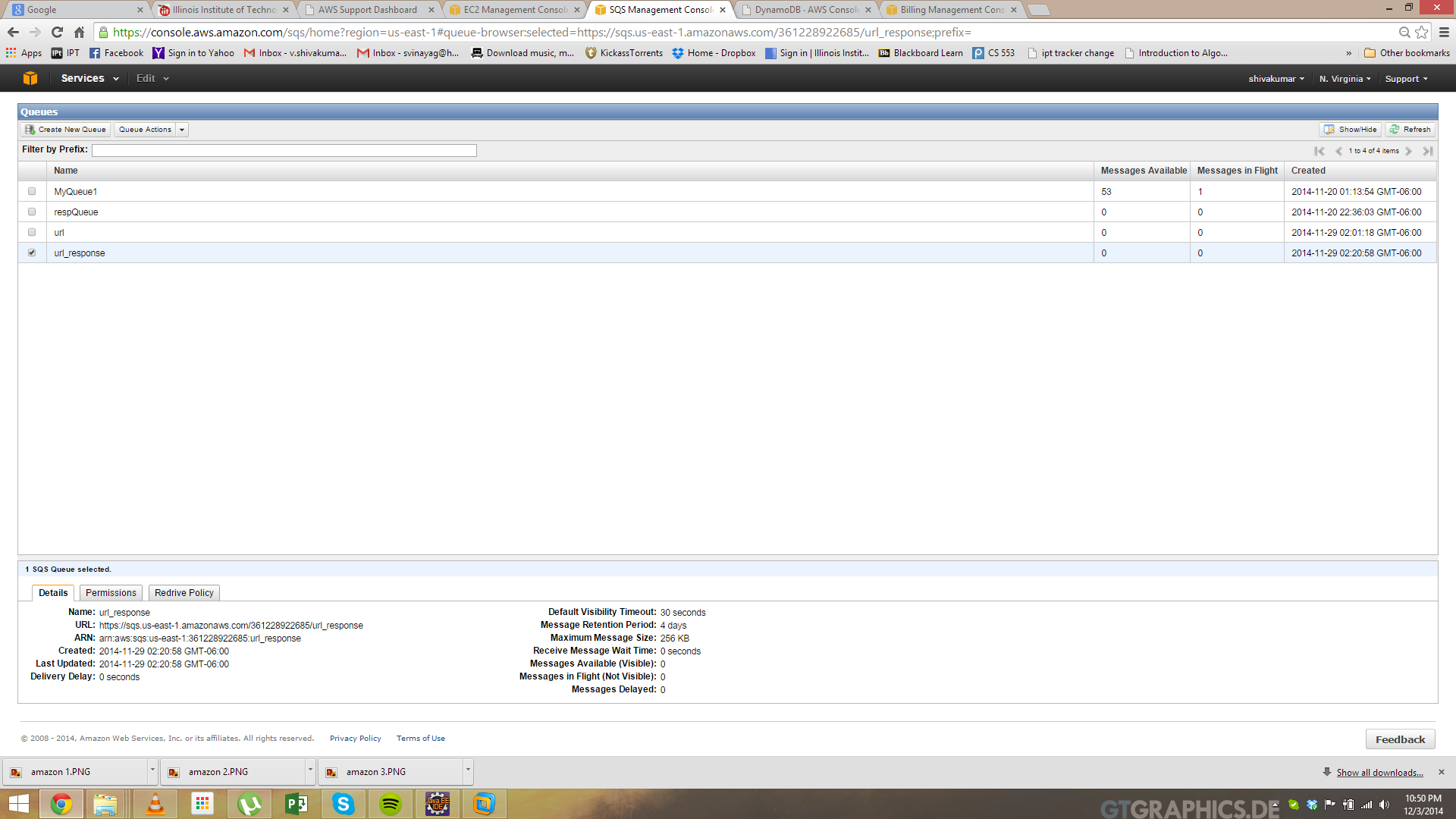




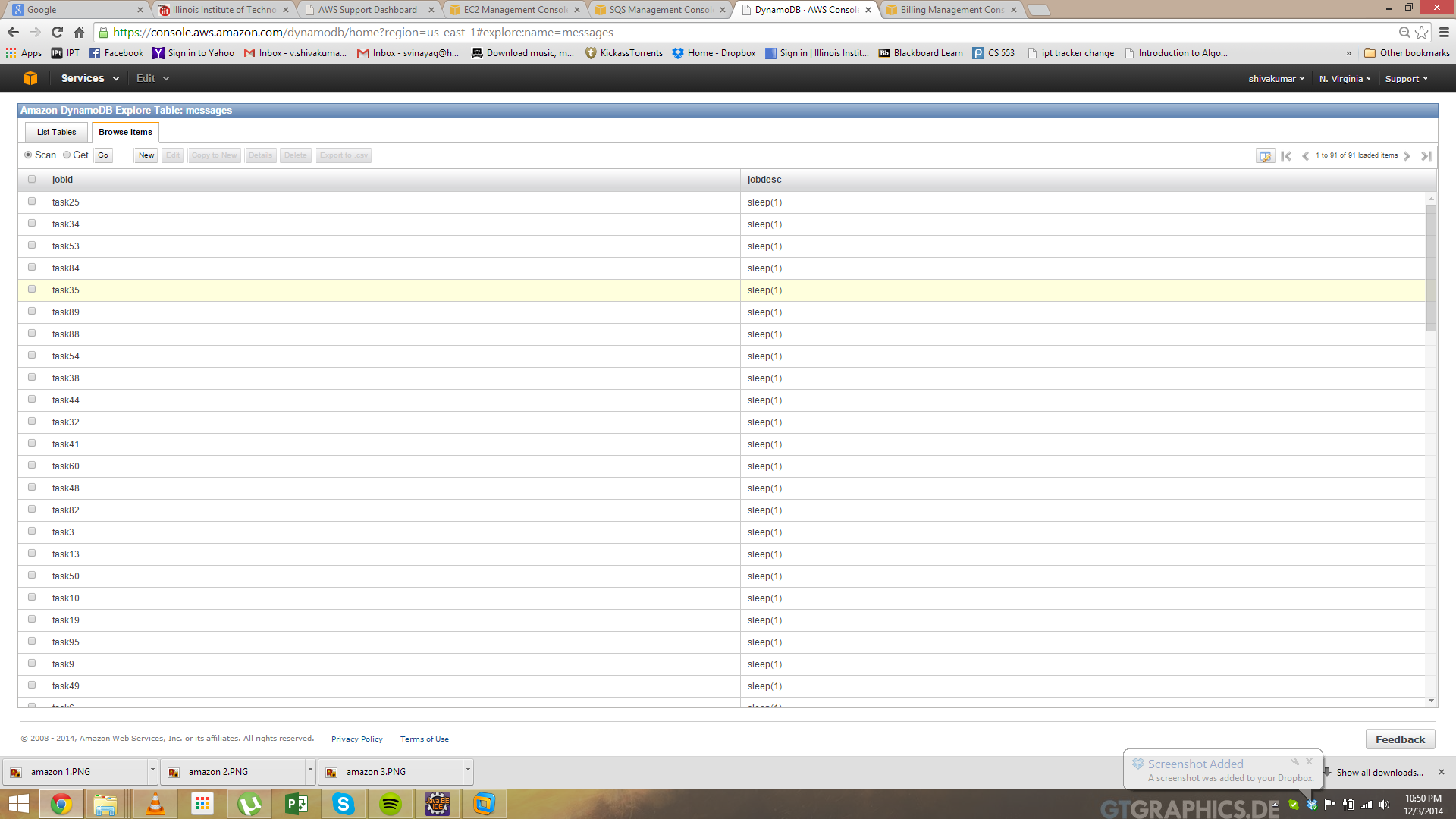
Client and server running plus static provisioned stopped machines:



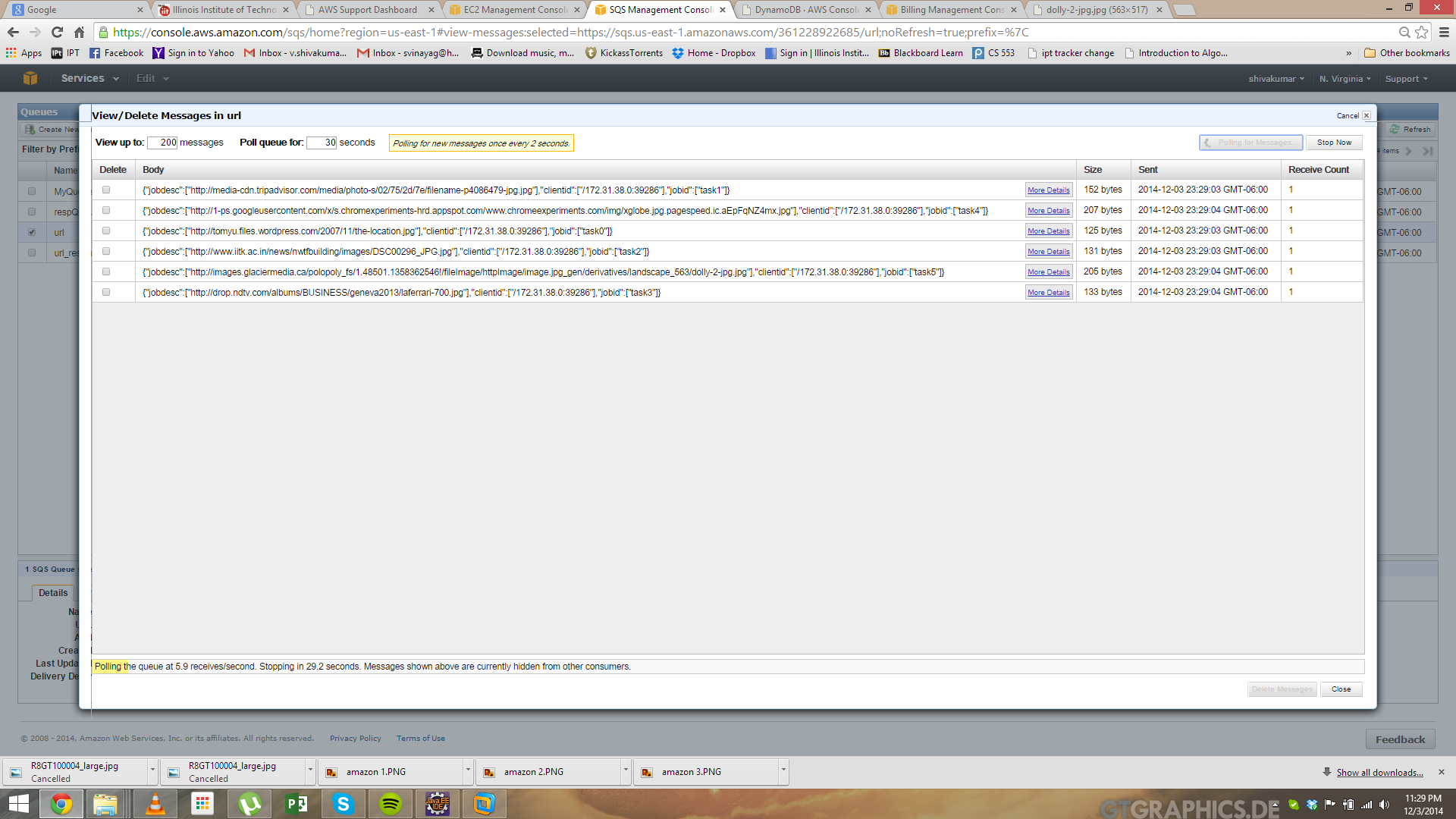
SQS view:



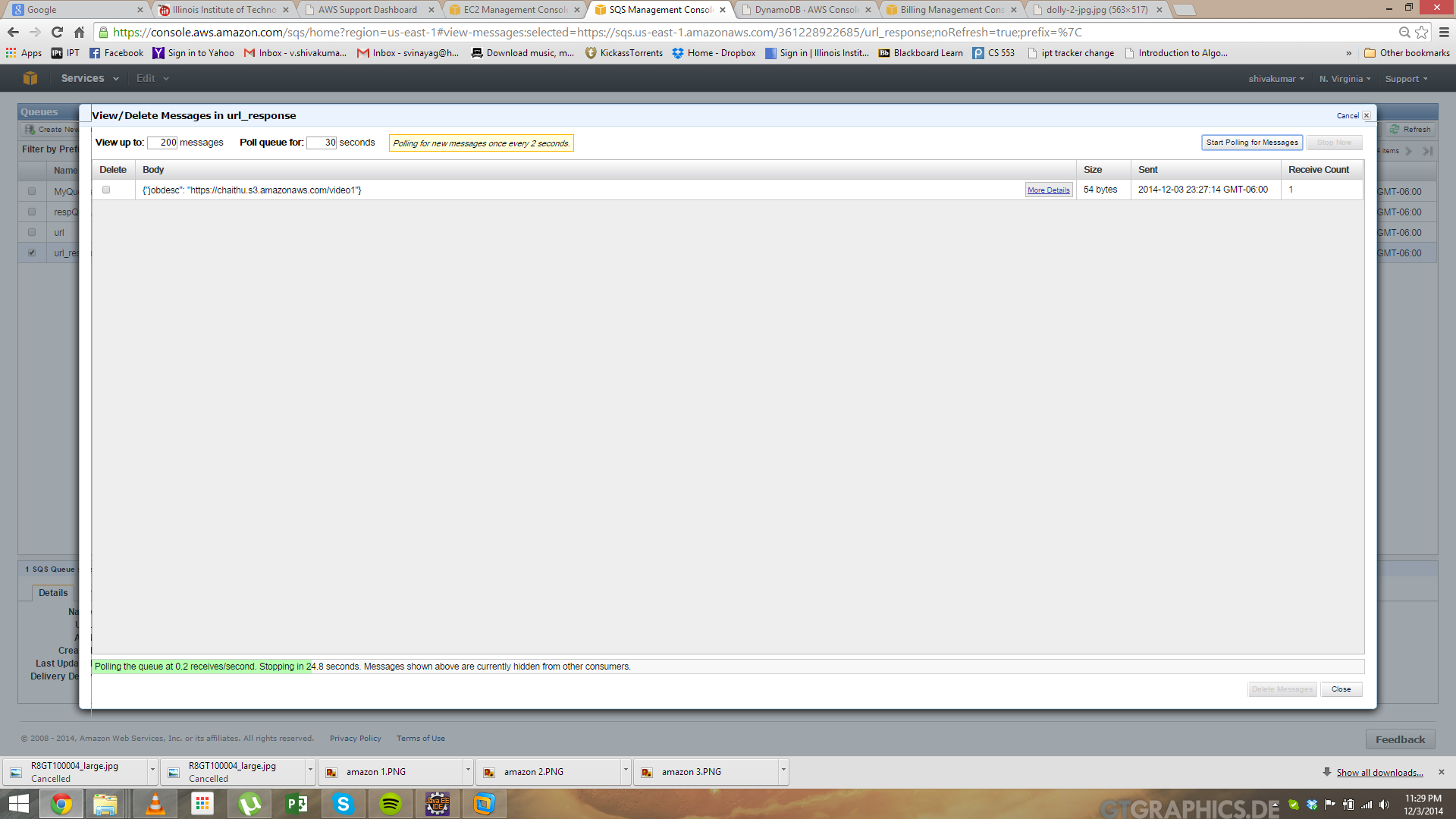
Dynamo DB view:



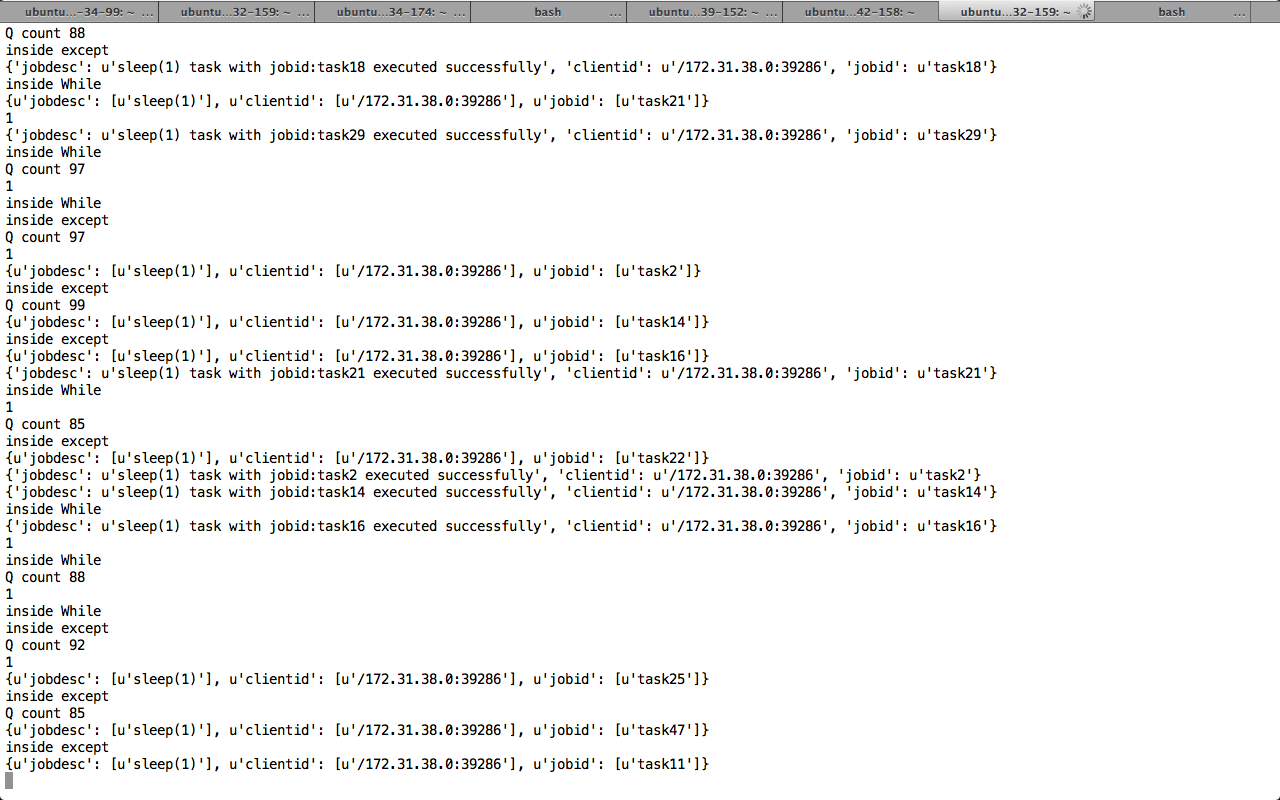
Animoto Url send queue view:

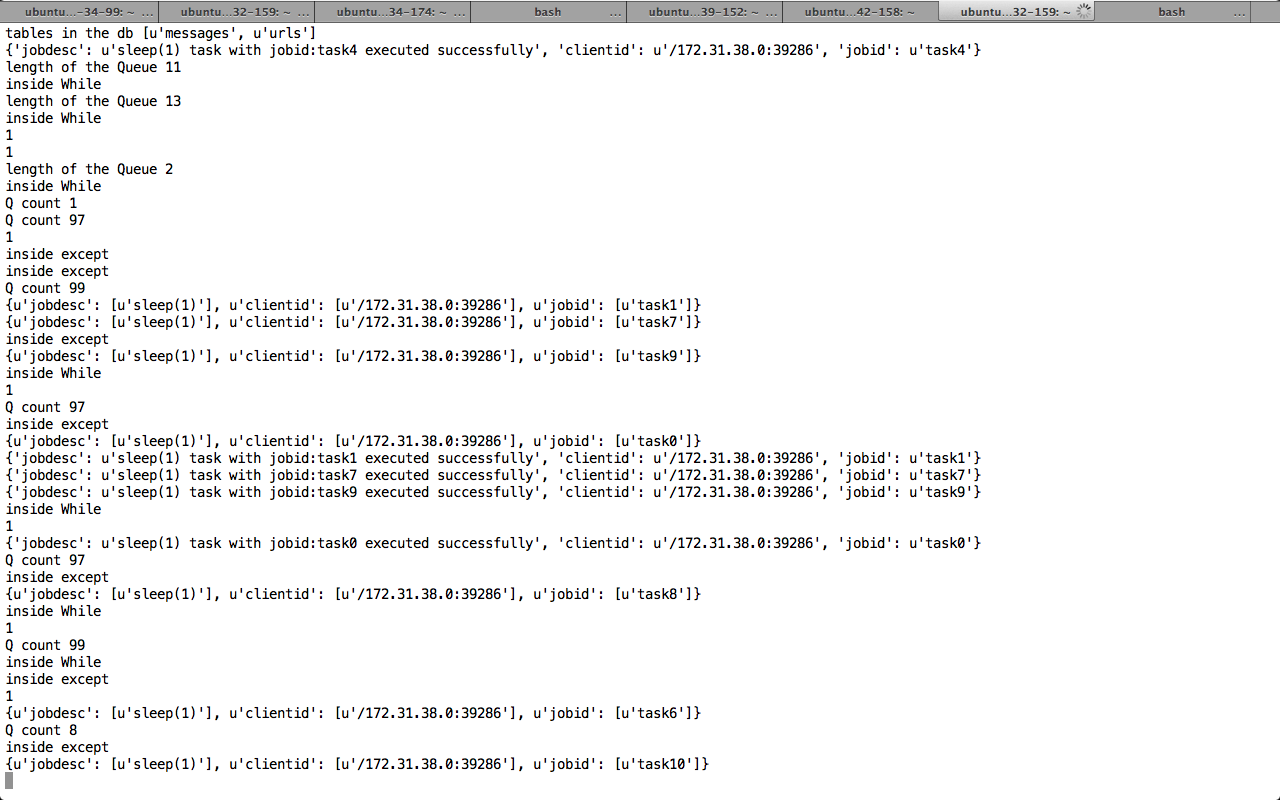


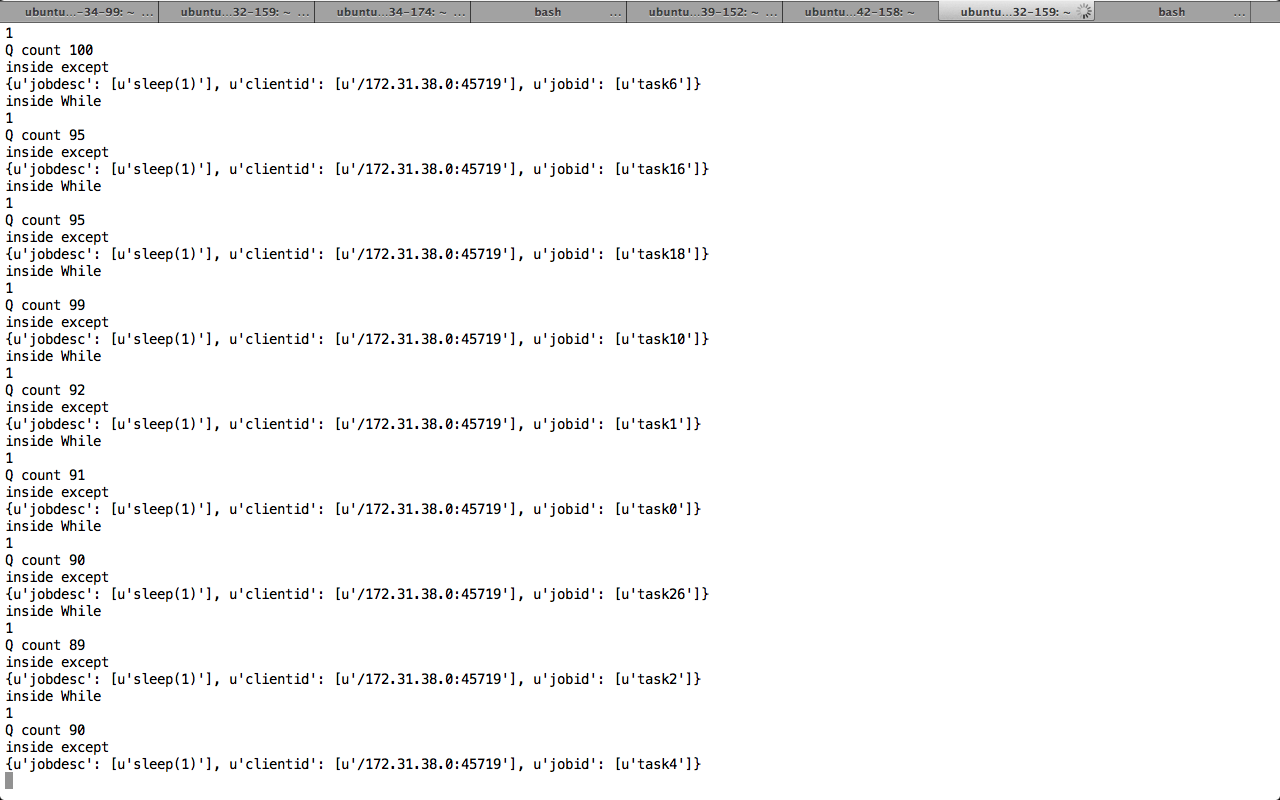
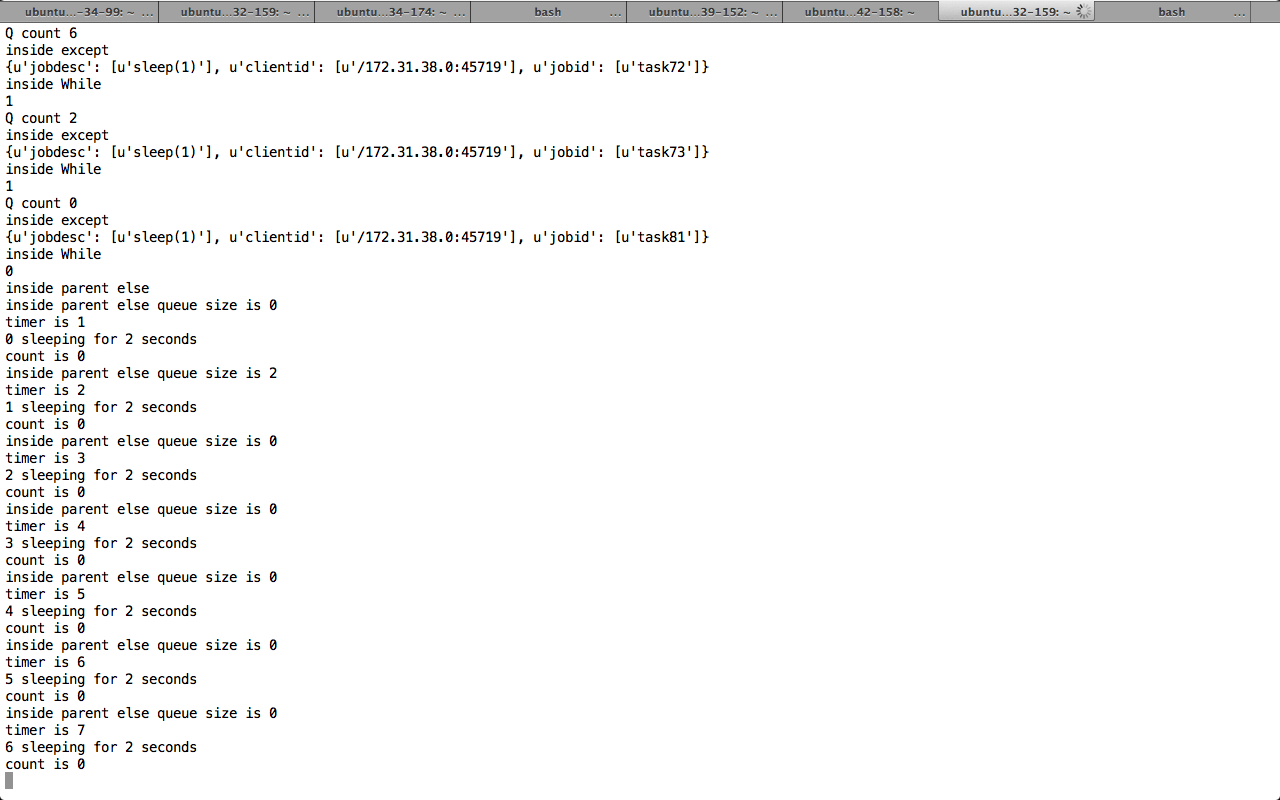
Animoto response queue view:



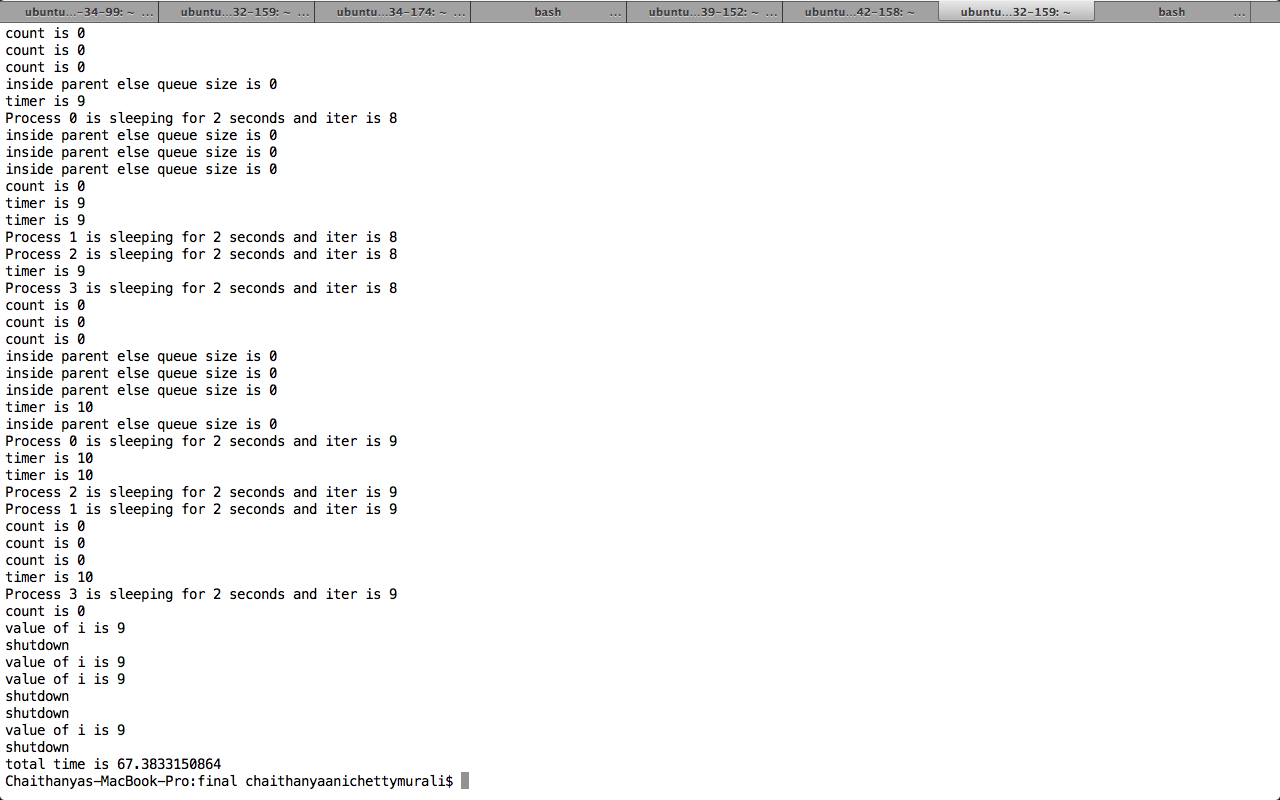
**Remote Worker Views:**

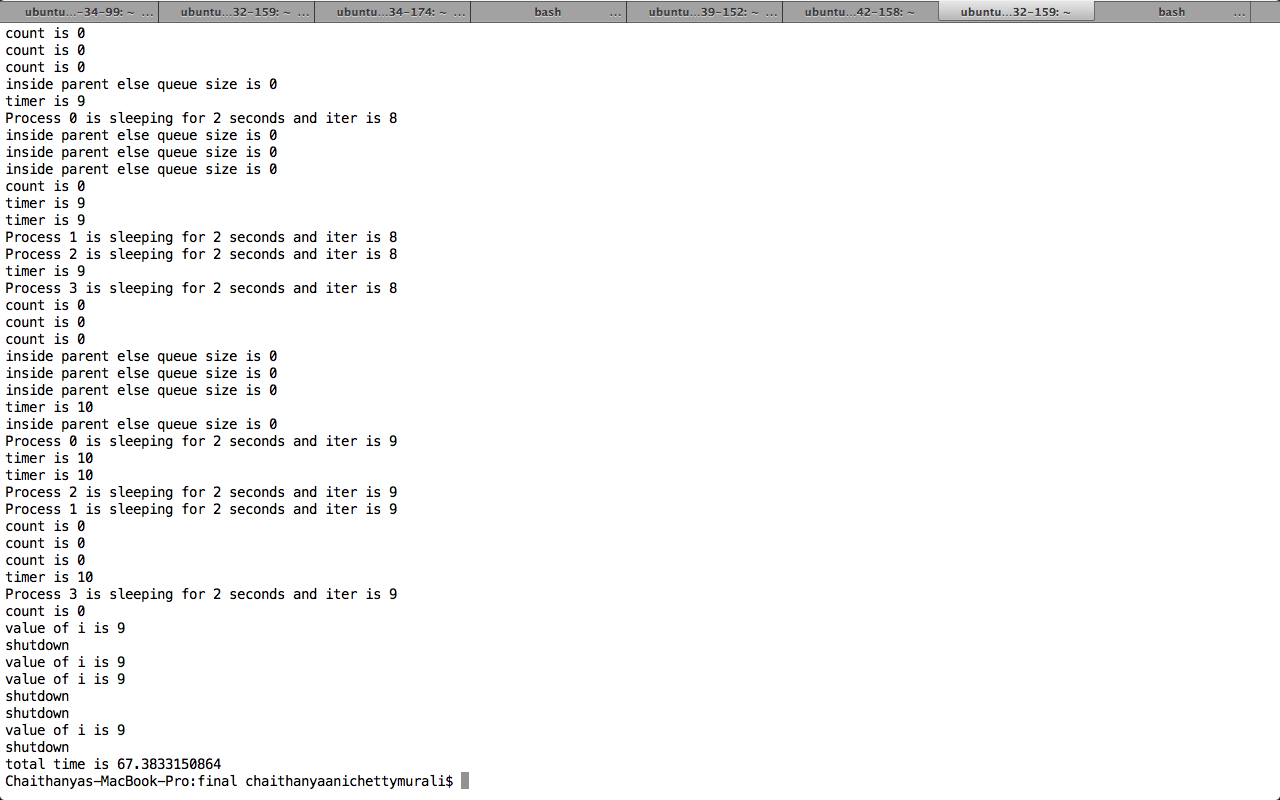




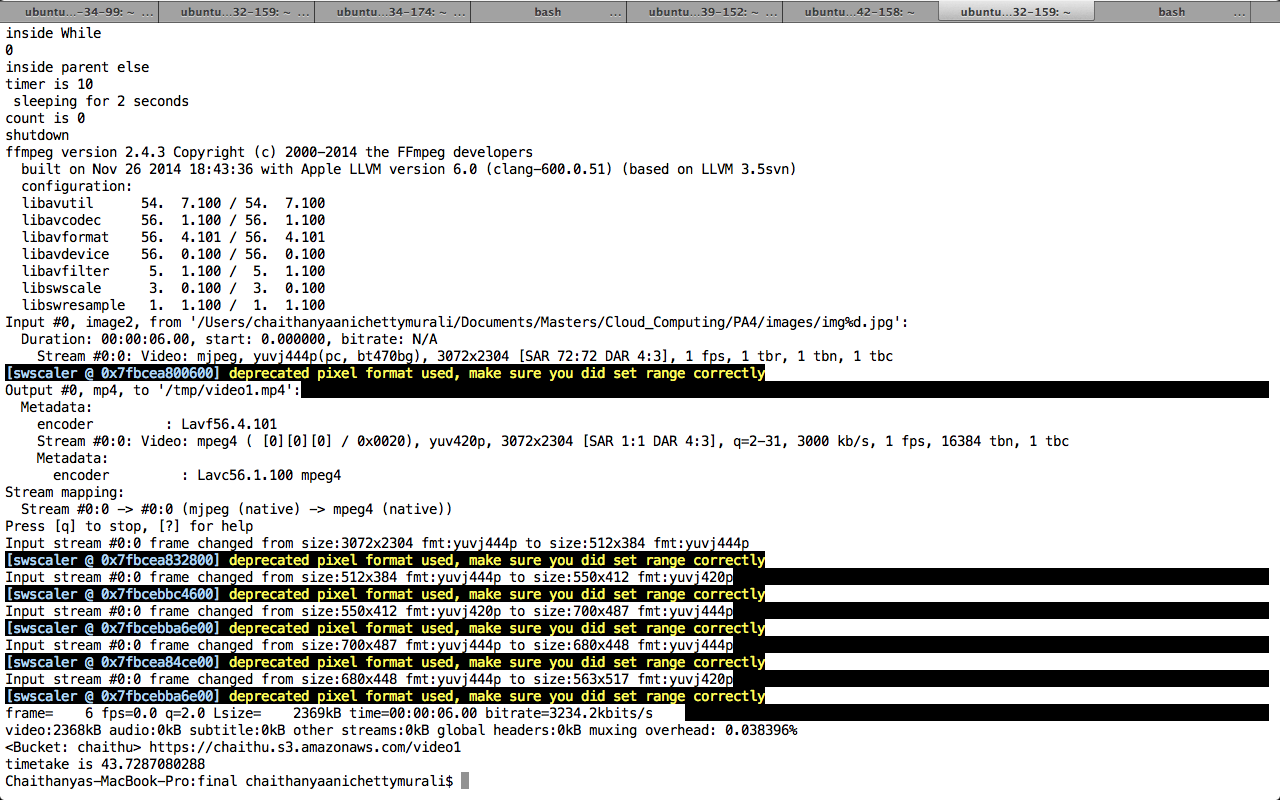


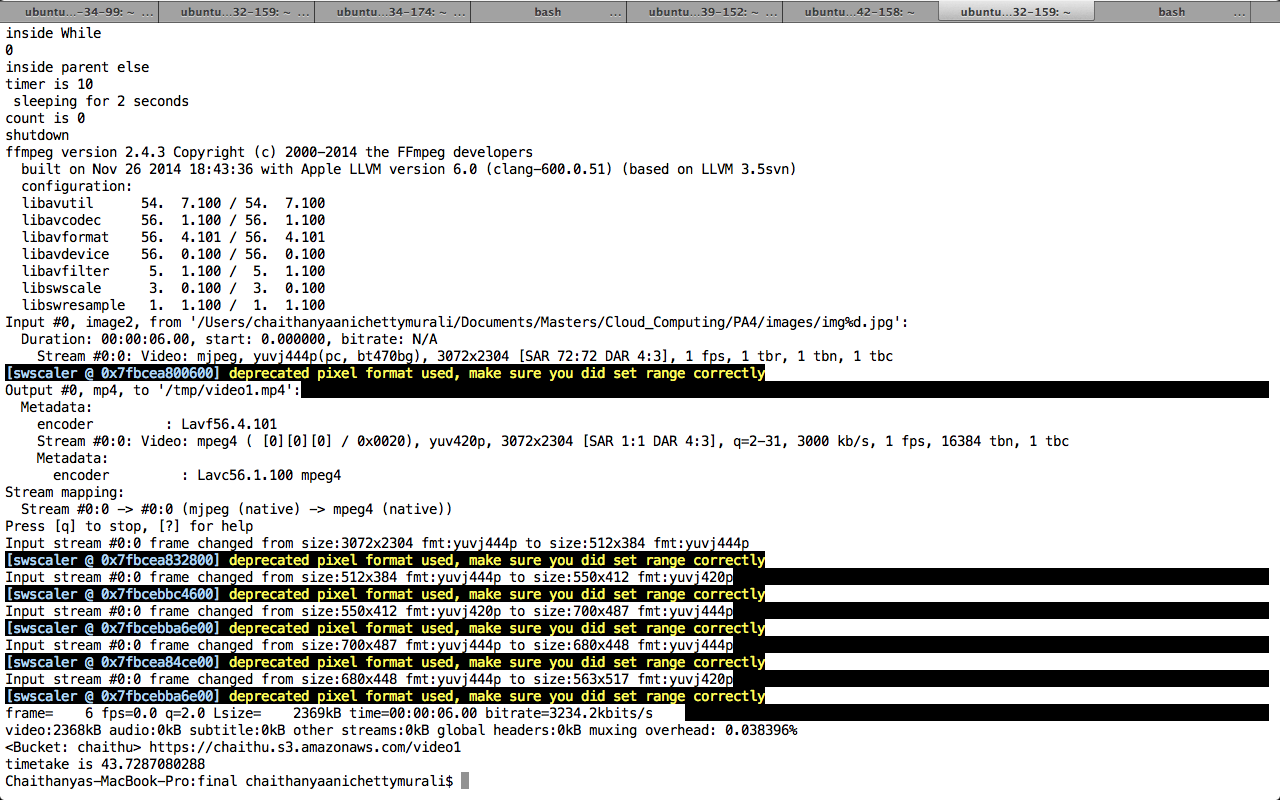
Remote working with threading:





Animoto working view:





Conclusion:

Hence the parts:

Client with batching

Scheduler with batching

Static scheduler

Dynamic Scheduler

Local Worker

Remote Workers with sleep time dynamic

Remote Worker with threading

Remote worker with normal Animoto

Remote worker with Animoto 1080 p

Are done and the screens shots and results are provided.