CS5412 Cloud Computing, Assignment 4
Name: Prashanth Basappa NetID: pb 476

Objective

In this assignment, we were asked to port project 3 on a EC2 instance and vary the number of clients to access the WCF service running on the cloud and observe the variation in results.

Design:

The entities that act as group members may be mobile devices, tablets etc. To handle this collection I have used nested dictionaries which are available in C#. The primary key for the dictionary will be a unique ID (e.g., name of the person, or the license plate of her car etc.). The primary dictionary value will be the nested dictionary which contain a set of associated attributes related to that particular entity.

Each group is typically associated with some user-defined class (here it is the collection of entities) that uses the group, and that class will often have a state that is partly replicated and partly local to each member. When the process is a full-fledged member, there are operations to multicast into the group, query it, update changes etc. The **multicast operations** are for Update and lookup using the delegate handlers in Isis2. Since the EC2 instance cannot be used for IP multicast, we will have to communicate using UDP which ISIS has a function: **ISIS_UNICAST_ONLY**, the system will just map all multicasts to UDP sends.

A get request can query an attribute and retrieve the primary key in the dictionary i.e., entity name and a set request can set new attributes or change the value of existing attributes. If it is a multiple attribute, we pass a list instead of a string. Hence, the dictionary will take in a list as the value.

I am using the OrderedSend to call the Set requests, and the Get requests are purely local because there is only server and Ordered Send will make sure the dictionary is updated and both are called randomly as required by various test cases. The **concurrency handling** is achieved using **semaphore locking**. I have used 3 semaphores (go, db_full and db_lock) The Get operation are locked using a semaphore (db_lock) for a particular entity from other set operations and avoiding a false read. The OrderedSend guarantees that all the replicas will come up with the same order of executing the commands. Isis2 helps in solving problems involving data replication within “groups” of programs.

ISIS_Environment Variables:

With the Isis2 OOB layer we can efficiently manage and modify the file replication patterns within the members of a group at very high speeds, shifting data, adding replicas, or deleting them, as frequently and concurrently. **The OOB layer is designed to handle large numbers of OOB files, and used highly efficient asynchronous messaging to transfer the data.** It then automatically checks for any chunks that may have been dropped and resends those, thereby ensuring that every recipient receives exactly one copy of each file.

ISIS_PORTNOp function opens a particular port on the service which was deployed on the cloud.

ISIS_HOSTS is used to add multiple ISIS servers are running on multiple EC2 instances.

Each iteration, each client will make a random decision to issue a read-only or an update request using the **Random class** available in C#, with the ratio of reads to updates hardcoded. **Clients will use only one thread and it will not limit the request because it is already taken care by the round trip time of the request i.e., the client will not proceed to the next request until the server has responded to the current request. The network I/O will bottleneck the requests on its own.** Also, my ISP may limit my bandwidth to limit my requests to make sure of net neutrality. As my EC2 instance is in Oregon, the RTT was observed to be ~0.5 second.

EC2 Setup:

Region: US West(Oregon)

I created a EC2 instance running Windows and RDPd into it. I **disabled the firewall** and open up the ports in the security group and I was able to communicate my clients from my local machine to the WCF service running on the cloud.

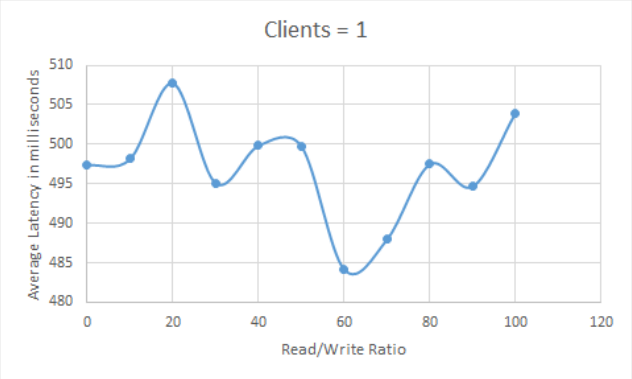
Graphs Observed:

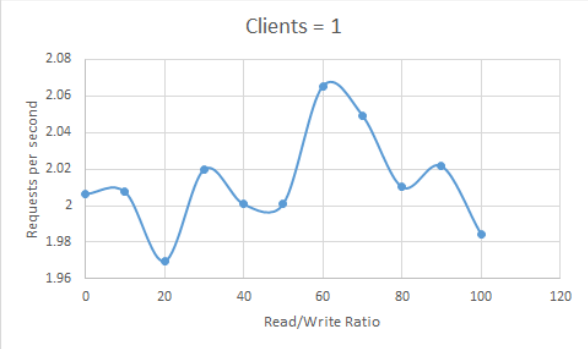
The following graphs show for the number of clients ran i.e., 1,2,5,10. Each case is run for a batch of 100 requests divided among GET and SET requests randomly called by the client. I have used Microsoft's StopWatch datatype to measure the delays in milliseconds(mS).

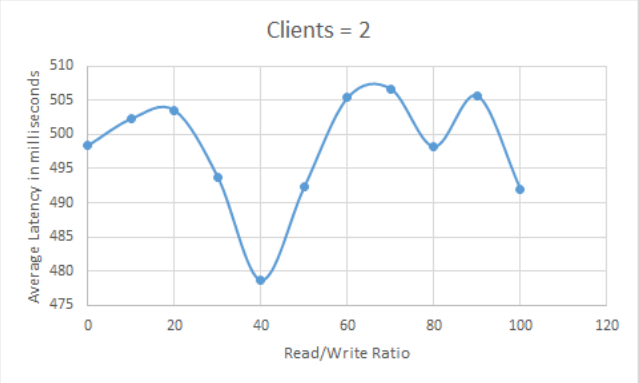
X-axis = the batch size and ratio of Read:Write requests

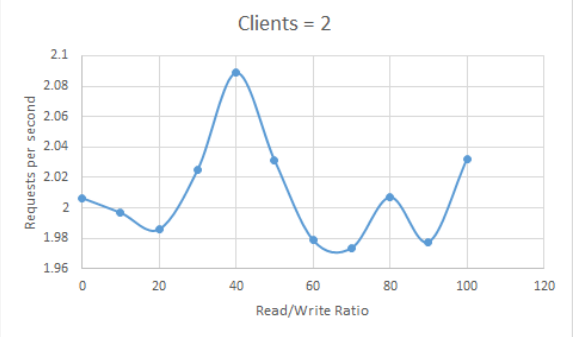
Y-axis = the average delay in microseconds (uS) and the number of requests per second.

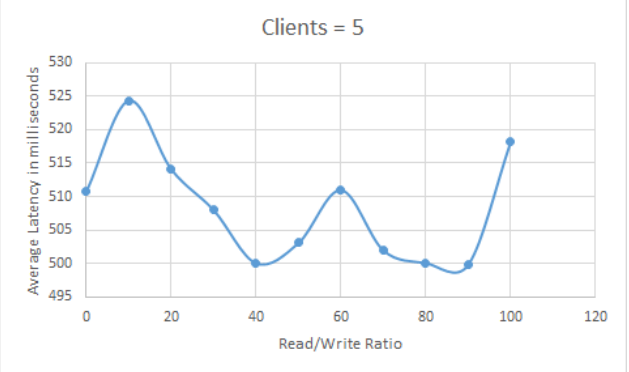
I ran the experiment for two criteria per read/write ratio: The **Average latency(in mS)** and **requests/sec** the service can handle.

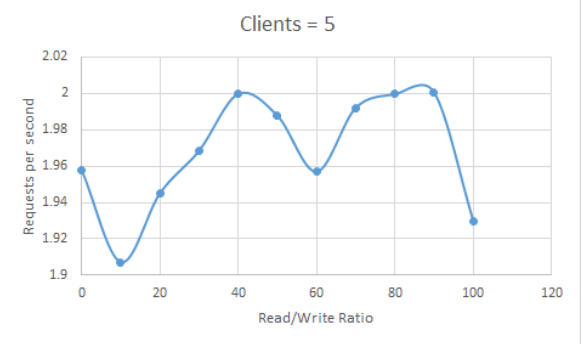


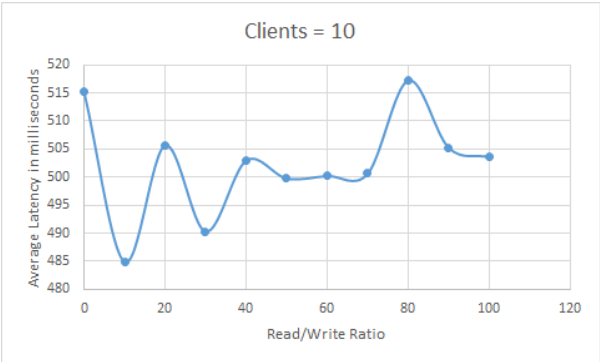


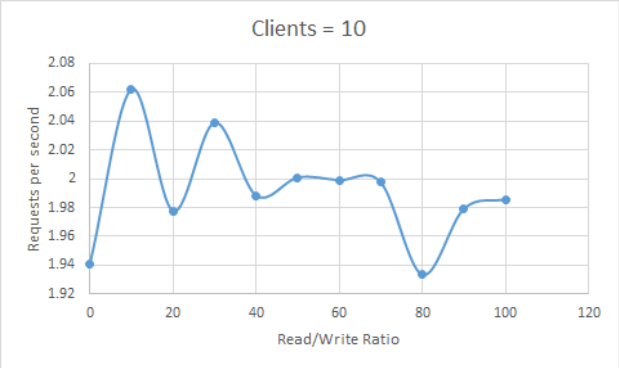












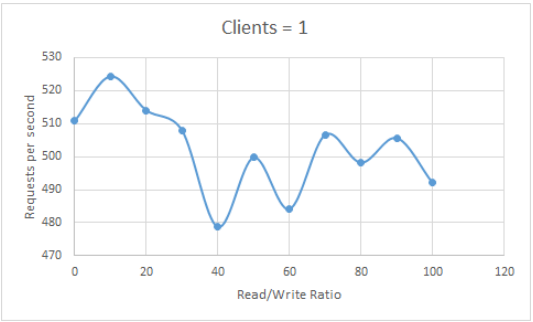
Explanation:

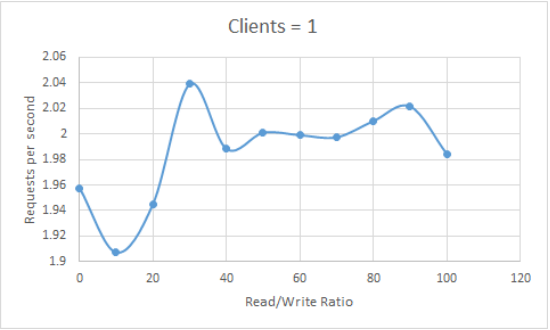
The graphs I observed for both latency and req/sec on teh and on the server were similar throughout. It had a on an average of ~490 mS and 2 req/sec. The constant latency can be explained by the fact that there is only **one server on the cloud and there is only one member service and hence all reads and writes will take the same time irrespective of the number of clients. There is no worry of maintaining consistent states across multiple servers and updates are not needed for a single member.**

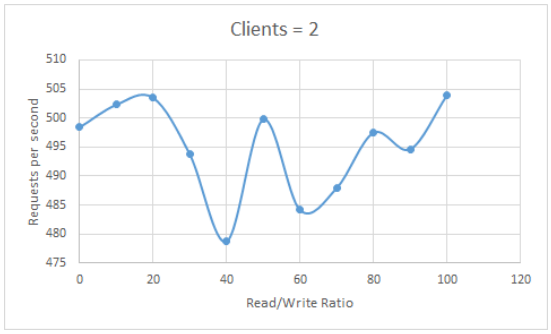
Extra Credit:

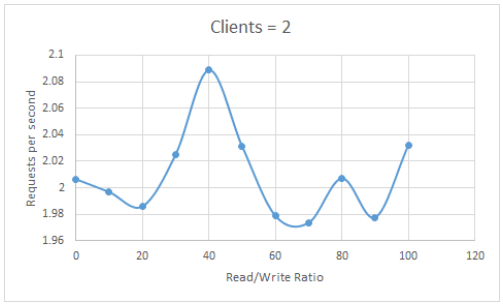
For extra credit, I set up two EC2 Microsoft instances and running two Isis2 members on the same group. While I send OrderedSend to one WCF service on one EC2 instance, it sends a OrderedSend to itself and other member on the same Isis2 group.

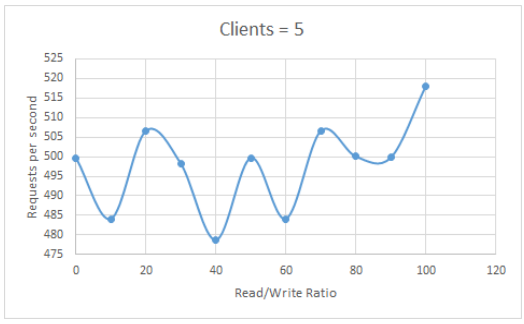
Graph Explanation: The graphs I observed were very similar to previous results where only one server ran on the cloud. Ideally, for two servers on different machines OrderedSend must have taken more time to maintain consistency across servers. This may be because the servers are on the same rack in my EC2 region i.e., in Oregon.
Following are the results:

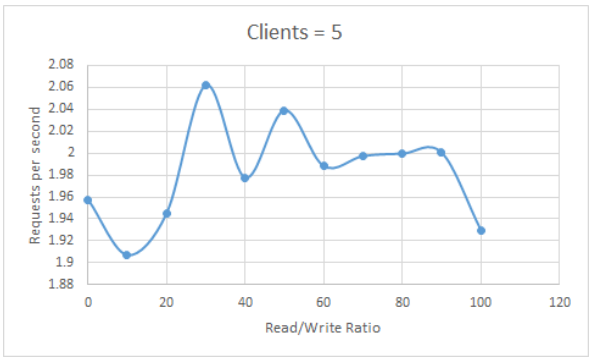


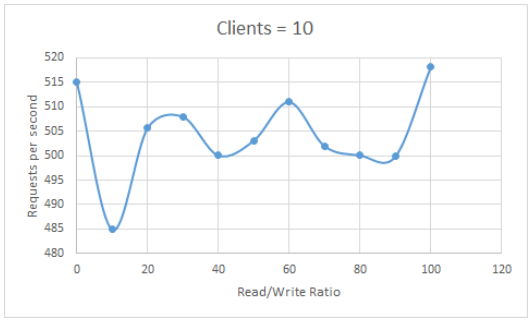


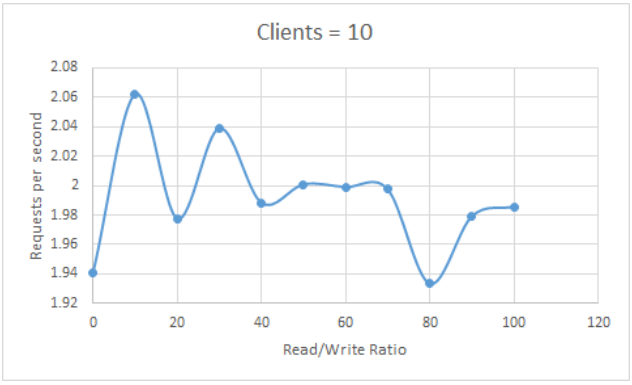




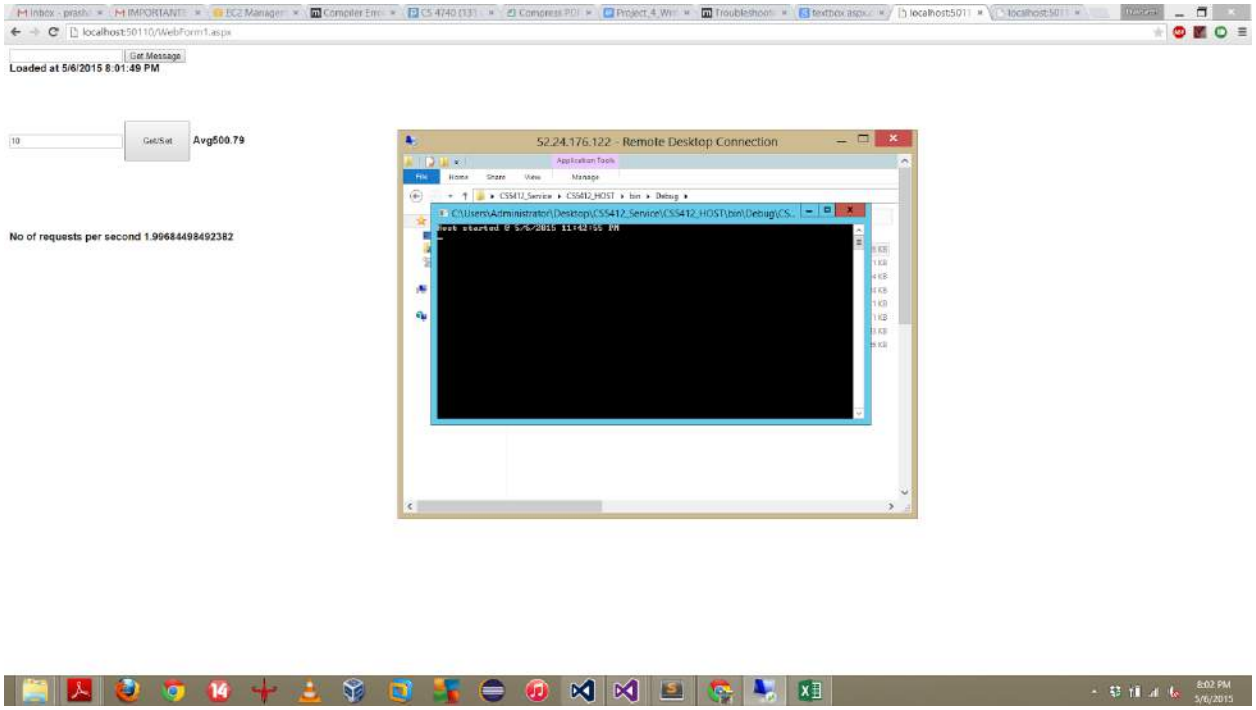




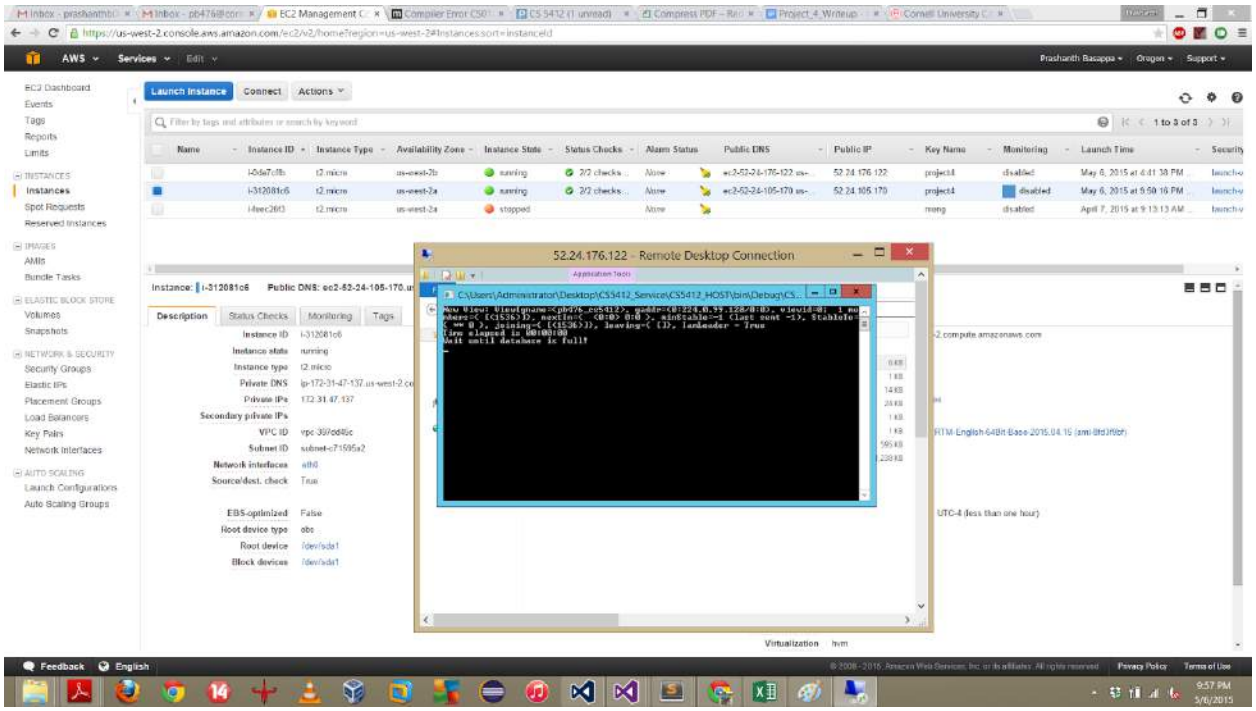




Screenshots:



In this screenshot, I am able to to make two clients query my WCF service which is running on a EC2 instance.



In this screenshot, I have made Isis communicate with the other Isis member running on another Ec2 instance.

References:

- Watched a youtube tutorial video to set up a WCF service and client.

Refer: https://www.youtube.com/watch?v=UcmD1SfIayM

- https://msdn.microsoft.com/en-us/library/1aey0kb6(v=vs.90).aspx
- AWS documentation on .Net application.
- https://msdn.microsoft.com/en-us/library/s35hcfh7.aspx