CS425: Distributed Systems – MP4 Report

Gopalakrishna Holla V (hollava2)
Alok Tiagi (tiagi2)

Design and algorithm

Upgrades to key value store

The key value store maintains an extra flag per key-value pair indicating whether it is the owner or the replica of the key-value pair.

Replication

Our replication strategy is passive. Carrying on from the system built in MP3, each key has an owner based on the hash of the key (and the hashes of the machines on the ring). The owner is responsible for maintaining copies of the keys it owns in its two immediate successors.

Replication handling by the owner

When the owner of a key receives a command through the routing process, before performing the command on the local key value store, it sends "force" commands to its replicas. These "force" commands indicate to a receiving server that the command does not need to be routed and is meant to be applied on the local key value store.

Event handling

Join

When a machine joins, exactly three machines take action. The first and second predecessors of the new machine replicate their keys on the new machine. The successor of the new machine transfers some of the keys that it is no longer owner for (since it belongs to the new machine) to the new machine and also deletes some of the keys that it is no longer replica for.

Failure

When a machine fails, exactly three machines take action. The first and second predecessors of the machine have just lost a replica; hence they re-replicate their keys on their new second successor. The successor of the failed machine now owns a few of the keys it previously replicated (the keys owned by the failed machine) so it identifies those keys, marks them as owned and replicates them on its two immediate successors.

(Leave is just a special case of failure where the system knows beforehand that a machine will leave. But the reorganization is handled in the same way as it would have been had the leaving machine failed)

Handling of two simultaneous failures

Since we maintain two replicas for each key (so each key value pair exists on three machines in the system; the owner and two replicas), the system can tolerate two simultaneous failures and re-replicates data as soon as failure is detected.

Concurrency

The client can specify the consistency level by indicating it in the command (o-insert means insert with consistency level of ONE, q-insert means insert with consistency level of QUORUM and a-insert means insert with consistency level of ALL). The "force" commands sent by the primary owner of the key to the replicas are threaded i.e the

command is sent to each replica via a dedicated thread acting as a client. This thread also receives the reply from the replica. If the consistency level is ONE, the primary does not wait for any thread. If the consistency level is QUORUM, the primary owner waits for any one of the two threads and compares the reply received with its own reply. If the consistency level is ALL, the primary owner waits for both the threads and compares all three replies to make sure that they are same. If not, it sends the client an agreement failure reply.

Last writer wins

We assume that the write order is the order in which the coordinator of a key (i.e. the owner of the key) receives the write commands. Since the owner executes any request it receives in program order, the last writer always wins. Additionally, the owner sends commands to both replicas before performing the commands on the local store. This ensures that the replicas never miss a value that has been stored in the store of the owner.

Application

We made a movie lookup application on top of the key value store. Given a word, the application returns all the movies that contain the word.

After downloading the movie list, we pre processed the data and indexed it. We picked out all the unique words in the data and designated them as keys. The value attached to each key is a list of indices. These indices are line numbers in the file that contains the movie list.

When a client looks up a particular key, the server that has a copy of the key examines the value attached. Using the list of line numbers, the server picks up the corresponding lines from the movie list (each server has a copy of the file) and sends it back to the client.

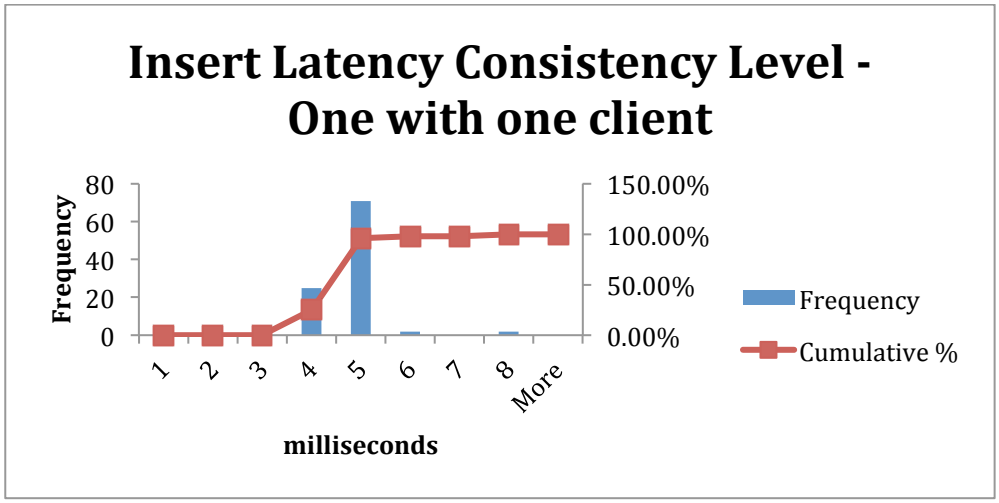
We used MP1 to track key value store operations and associated exceptions.

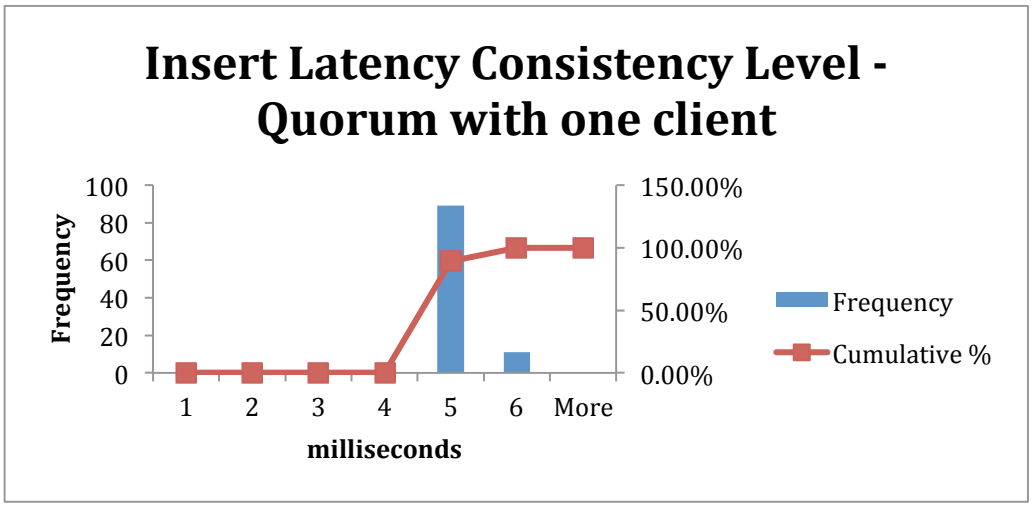
(Reading for lookup and insert latencies in extra credit part)

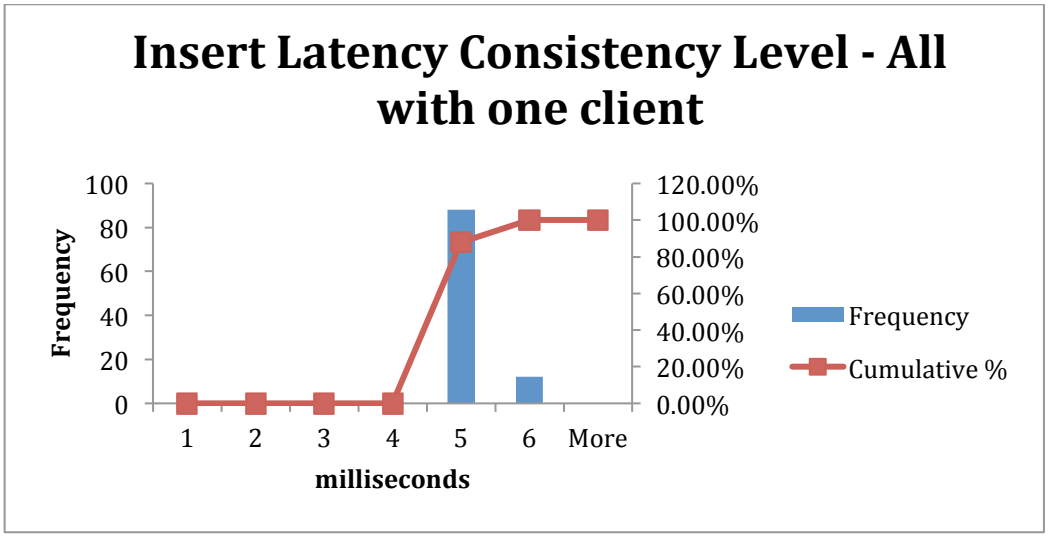
Extra credit report

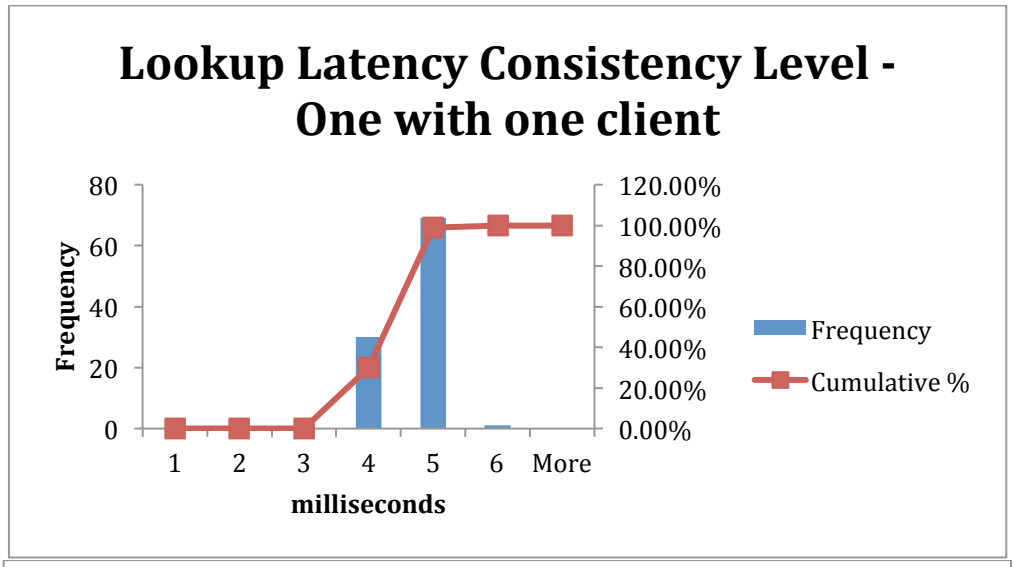
Method for taking readings for insert and lookup latencies

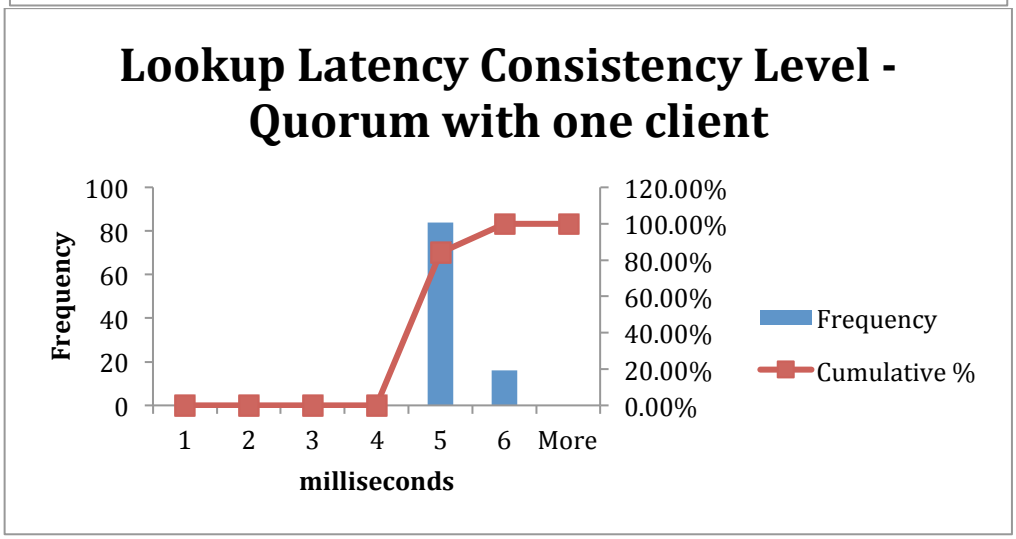
We ran a script that inserted 1000 keys into a system of 4 servers. Then we looked up/updated 100 keys and timed each of the operations.

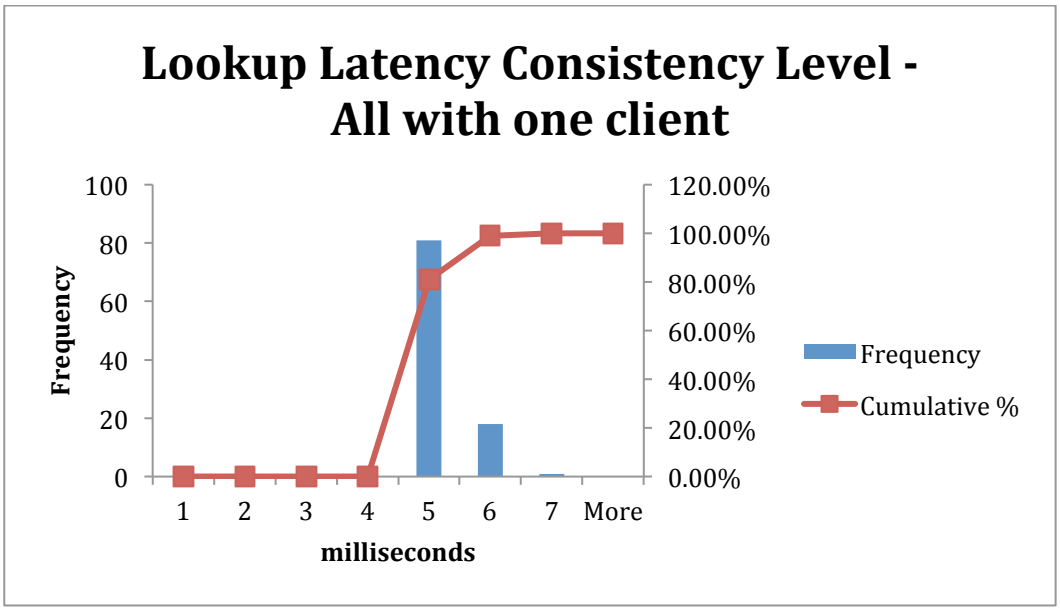







Lookup Latency Consistency Level - One with one client
Frequency
80
60
40
20
0
120.00%
100.00%
80.00%
60.00%
40.00%
20.00%
0.00%
1
2
3
4
5
6
More
milliseconds
Frequency
Cumulative %

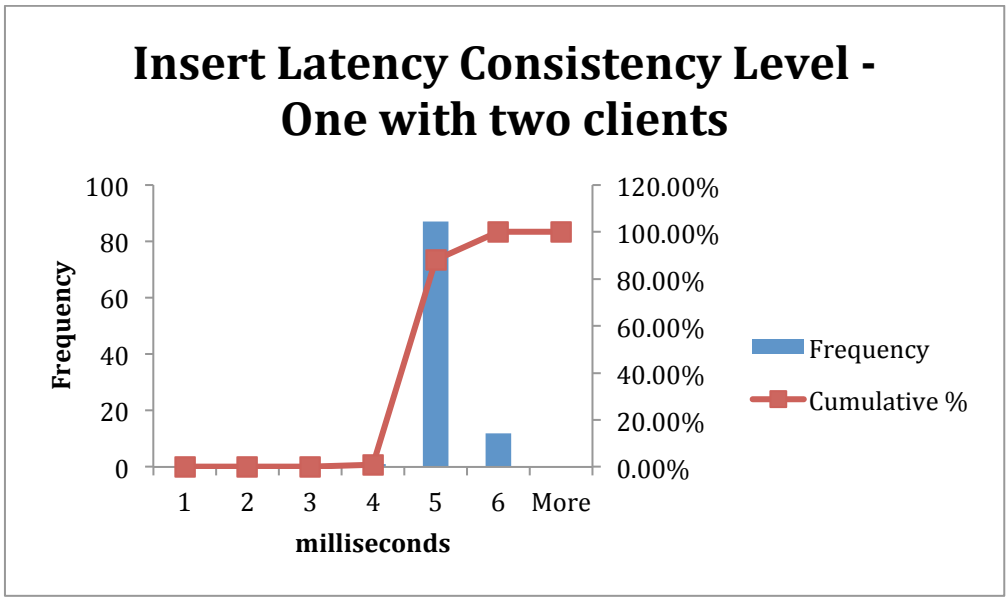

Lookup Latency Consistency Level - Quorum with one client
Frequency
100
80
60
40
20
0
120.00%
100.00%
80.00%
60.00%
40.00%
20.00%
0.00%
1
2
3
4
5
6
More
milliseconds
Frequency
Cumulative %

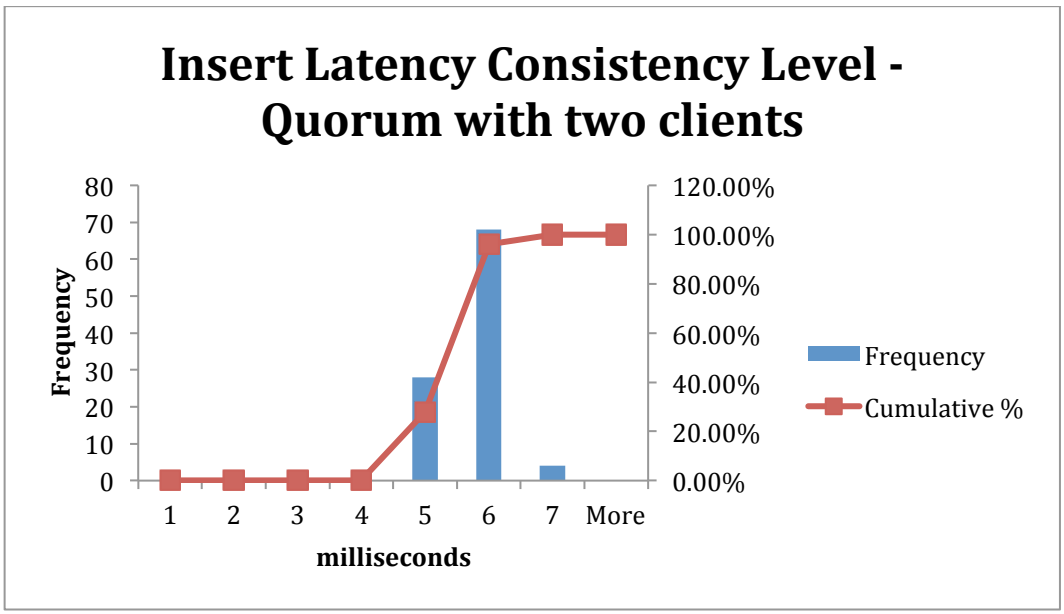

Lookup Latency Consistency Level - All with one client
Frequency
100
80
60
40
20
0
120.00%
100.00%
80.00%
60.00%
40.00%
20.00%
0.00%
1
2
3
4
5
6
7
More
milliseconds
Frequency
Cumulative %

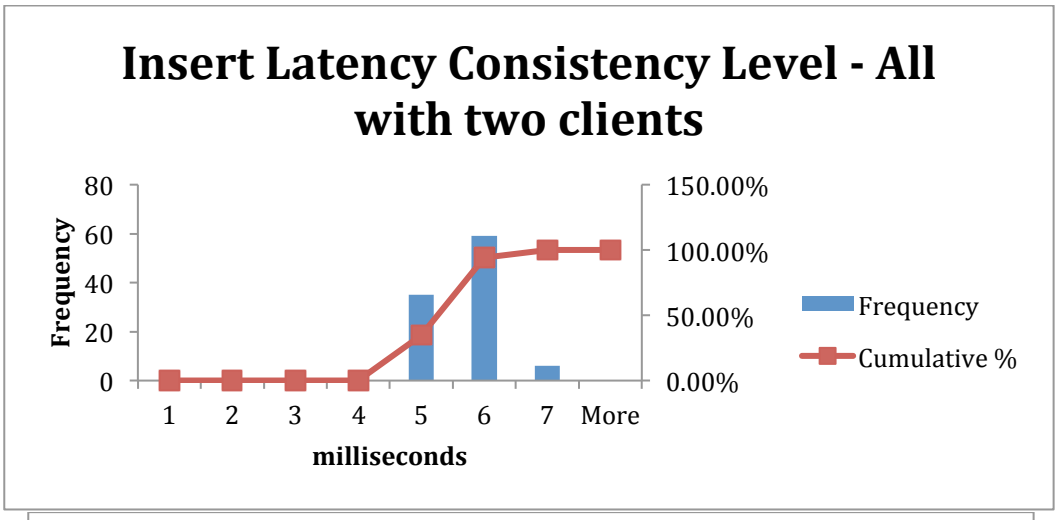
Discussion on insert and lookup latencies

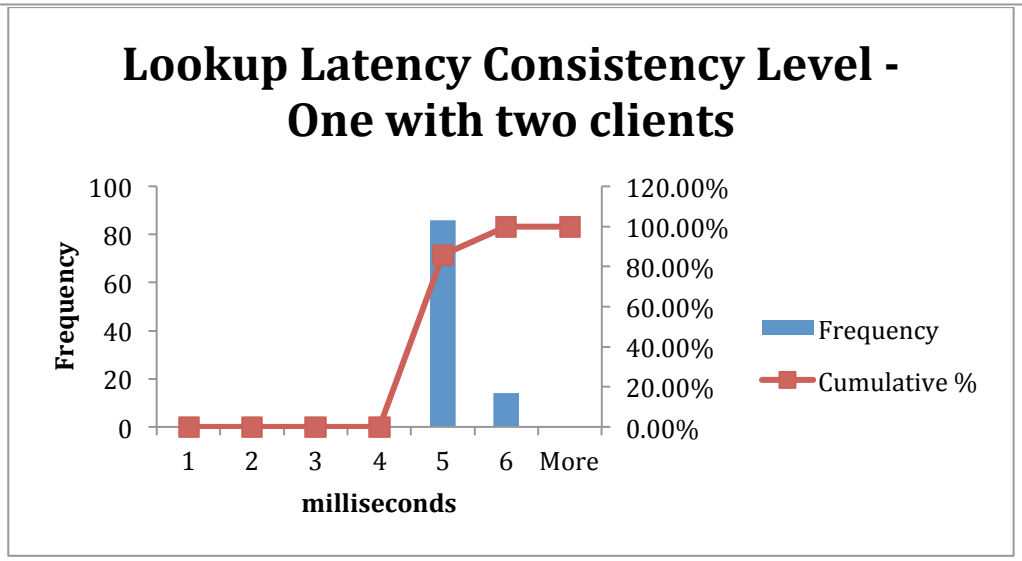
The difference in latencies between 'one' and 'quorum', though negligible (mostly less than 1 ms), is obvious from the CDF. This is because of the threaded approach of our design where the primary owner of the key threads the "force" commands' sending to its two replicas and does not wait for the threads to get back with a reply.

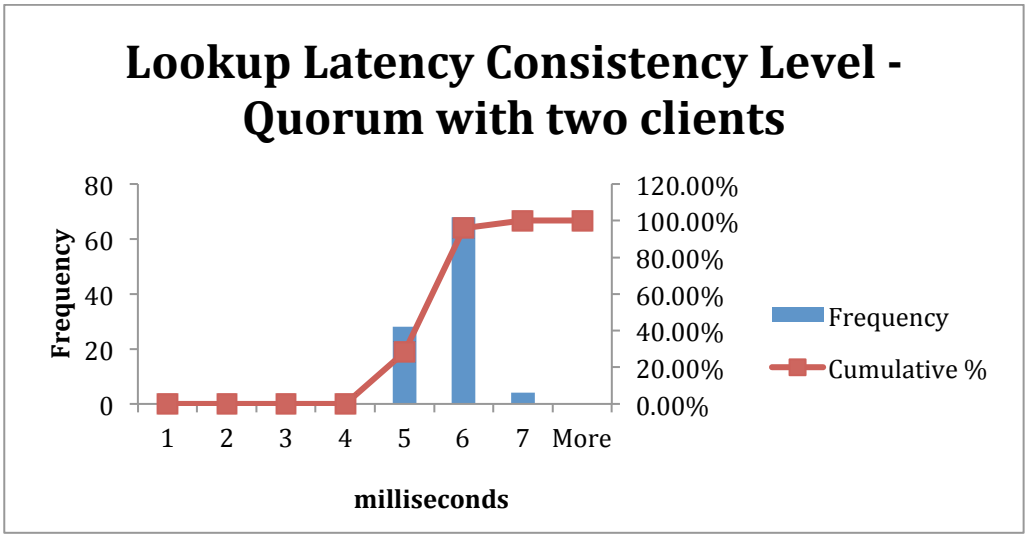
There is almost no difference between the 'quorum' and 'all' consistencies. This is also a result of our threaded approach. Since the requests are sent in parallel to both replicas, waiting for one or both to come back is not much different in terms of time. It is to be noted that this is only because the machines that we conducted the tests on are physically very close to each other. When a large physical distance separates replicas, quorum will be faster than all if at least one replica is much closer than the other.

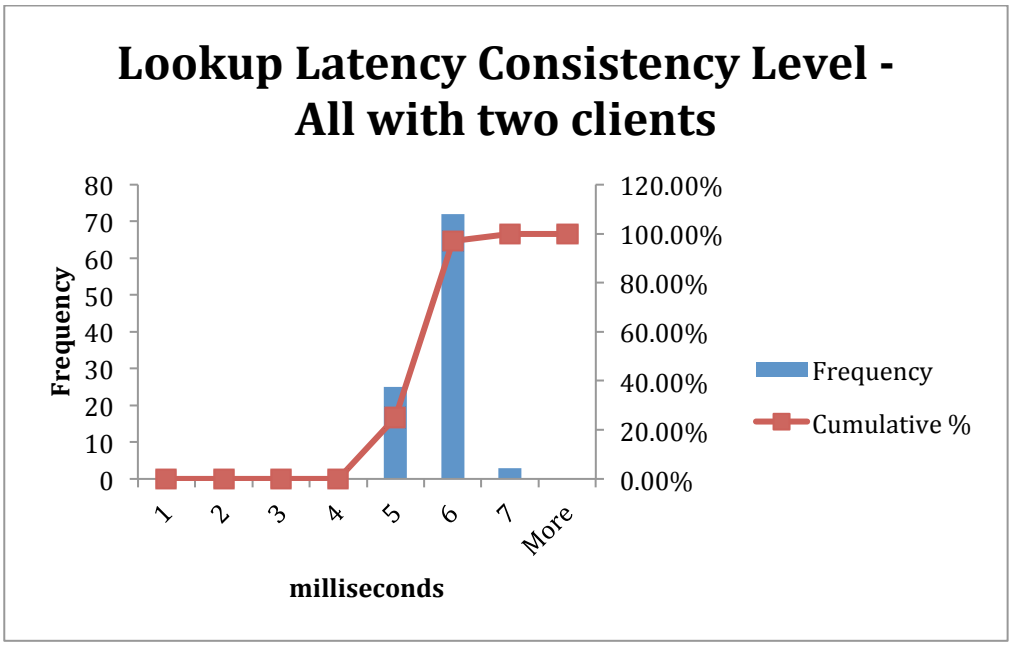





Insert Latency Consistency Level - All with two clients
Frequency
80
60
40
20
0
150.00%
100.00%
50.00%
0.00%
1
2
3
4
5
6
7
More
milliseconds
Frequency
Cumulative %


Lookup Latency Consistency Level - One with two clients
Frequency
100
80
60
40
20
0
120.00%
100.00%
80.00%
60.00%
40.00%
20.00%
0.00%
1
2
3
4
5
6
More
milliseconds
Frequency
Cumulative %


Lookup Latency Consistency Level - Quorum with two clients
Frequency
80
60
40
20
0
120.00%
100.00%
80.00%
60.00%
40.00%
20.00%
0.00%
1
2
3
4
5
6
7
More
milliseconds
Frequency
Cumulative %



Discussion of differences between one client and two clients

There is a slight increase in latency with two clients as compared to one client. This can probably be explained because of the slightly increased load on each server.

Additionally, our design locks the key value store before performing any write operations. So the lock contention due to multiple concurrent requests might have increased the latency of the commands.

Method for taking staleness readings

We ran two clients on one threaded program. One client regularly updated the value of a key (cyclically changed it from 1 to 10) and another client repeatedly read the value of the key. We monitored the timestamps and also monitored the output of the reads to make sure there were no repeats (which would indicate a stale value). We found no staleness in the readings.

Reasons for no staleness in our design

Because of our design, there will never be staleness in the reads. The owner of the key is responsible for processing all requests for the key (and it does this in the order it receives the requests). Hence a read for a key sent after a write for the same key will see exactly the value written by the write. If a read goes in before a write, it will see the value written by the previous write. Replicas are used only for fault tolerance and for maintaining consistency levels and do no process reads independently (they always receive lookup commands from the owner of the key, never directly from a client).