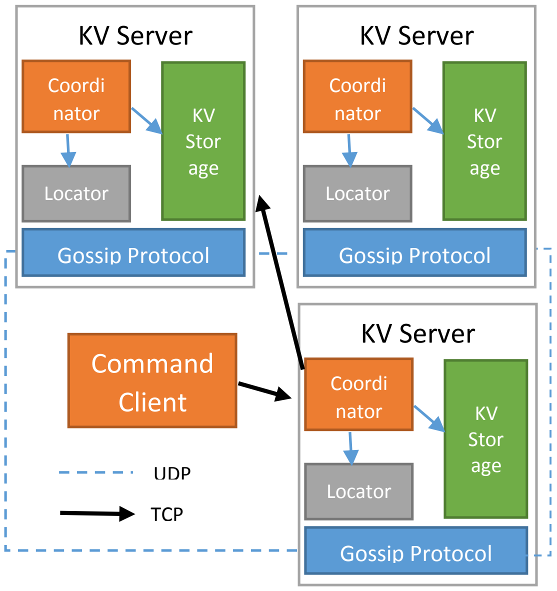
CS425 MP3 My Key-Value Store Report

Le Xu (lexu1), Wenting Wang (wwang84)

System design: In this assignment, we implemented a distributed key-value store system on the top of membership gossip protocol. The key value storage is distributed among different nodes based on a chord-like consistent hashing (which reduce the workload while members join or leave the group). The hashing information is stored in a component called Locator. The underlying gossip protocol keeps records of new node joining and node leaving, and updates the information in Locator to keep consistent.



When users input insert/update/delete/lookup commands in CommandClient, it connects to an arbitrary node and forwards requests to its coordinator component. The coordinator performs one of the two operations based on the result that whether the key of the operation is stored locally from locator:

1. Executes operations locally
2. Sends operations to a remote node who has the key

Logging system from MP1 (not pictured above) can be used to debug system by grep logs generated.

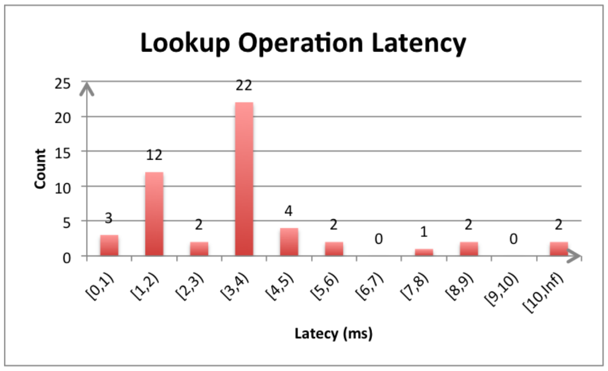
Algorithm: We choose Hash over Segment to assign nodes onto the ring because hashing works better than segments when system churning. Only a part of key/value pairs are affected in the neighbors of new joined/leaving node. In contrast, for segments approach, almost every node needs to migrate data to the new/leaving node. But the problem with hashing approach is load-imbalance. To address this problem, we use virtual nodes that minimize the impact of imbalance caused by randomness of hashing.

Experiments settings: 4 PC: 2 EWS Linux, 1 Mac PC, 1 Windows 8 PC

Measurement:

Lookup when the data structure is empty (milliseconds)

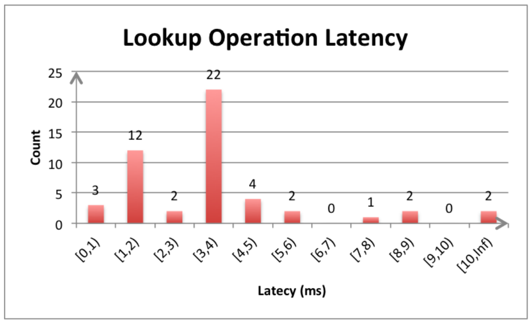
Average	Standard Deviation	Confidence Interval
27.56	174.54	48.38



We observed that most lookup operation (while the storage is empty) normally successfully finished within 0~4 milliseconds. For the keys being hashed to local storage especially, the lookup time can be even shorter (normally 1 milliseconds). However, while performing lookup for keys on remote server, the operation latency depends on the network connection (which is unpredictable) and can be large while connection is slow. The outliers also cause the average and standard deviation to be relatively large.

Lookup when the data structure is empty

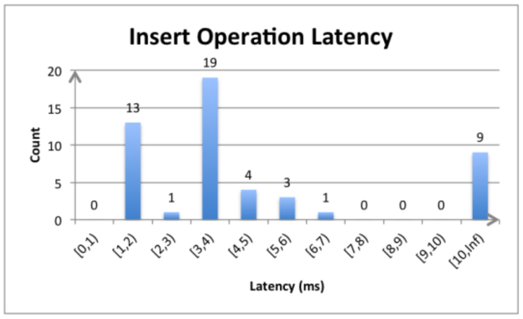
Average	Standard Deviation	Confidence Interval	Median
27.56	174.54	48.38	3



Lookup latency while the structure is empty: Most results are within 0~4 milliseconds. The lookup latencies for keys in local machine are small and stable (mostly 1ms) while for latencies for keys in remotes server are relatively unpredictable and results in large outlier (1237ms) depends on the network condition.

Insert when the data structure is empty

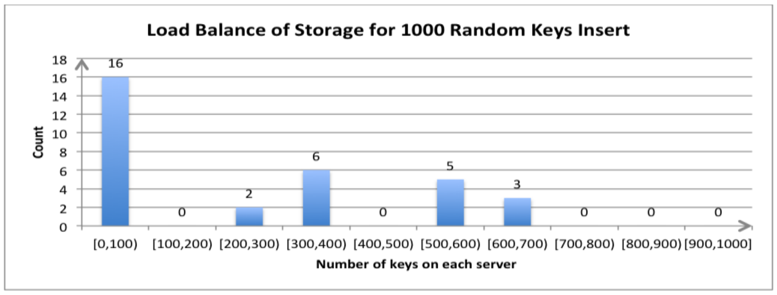
Average	Standard Deviation	Confidence Interval	Median
11.4	23.90	6.62	3



Insert latency while the structure is empty: Similar to the figure above, most requests can be finished within 4 milliseconds. However, we observed more requests takes longer to process (longer than 10ms)which indicates that generally insert operation has larger latency than lookup operation while the storage is empty.

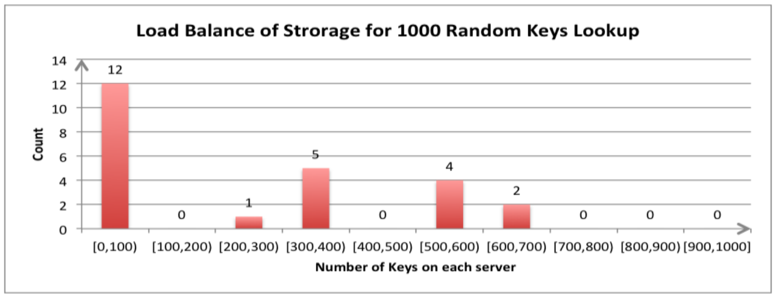
Load-balance of storage (1000keys)

Server	Average	SD	CI
Node1	319.125	25.78	19.10
Node2	578.875	26.28	19.47
Node3	82.25	3.92	2.90
Node4	19.125	6.73	4.98



Load-balance of lookup (1000 keys)

Server	Average	SD	CI
Node1	316.8	19.36	16.974
Node2	581.4	23.13	20.27
Node3	80.8	10.66	9.35
Node4	21	4.949	4.33



Since we chose to use chord-like consistent hashing (using server's ip address and port number), we noticed an expected drawback of this approach – some servers experience unbalancing data load while distributing keys (less than 100 keys or more than 500 keys). However, using consistent hash does significantly reduce the data migration while system churning. Since keys distribution of the group is static (depends on server's ip address and port number), load-balance of lookup/insert operation shows similar pattern as shown in the figures above.