An Empirical Evaluation of Distributed Key/Value Storage Systems

Instructor: Prof. Ioan Raicu
Group-01: Vaishnavi Manjunath (A20446043)
Kalpana Pratapaneni (A20448916)
Kavya Ravella (A20444960)

Introduction:

This report presents a brief discussion on three distributed key/value stores: Memcached, MongoDB, Redis and ZHT. **ZHT** aims to be a building block for future distributed systems, such as parallel and distributed file systems, distributed job management systems, and parallel programming systems. ZHT has several important features such as being light-weight, dynamically allowing nodes join and leave, fault tolerant through replication and by handling failures gracefully and efficiently propagating events throughout the system, a customizable consistent hashing function, supporting persistence for better recoverability in case of faults, scalable. **Memcached** is an in-memory key-value store for small chunks of arbitrary data (strings, objects) from the results of database calls, API calls, or page rendering. Memcached is simple yet powerful. Its simple design promotes quick deployment, ease of development, and solves many problems facing large data caches. **Mongo DB** has key features like ad hoc queries, indexing, replication, load balancing, map reduce and aggregation tools, schema-less database, high performance, stores files of any size and supports BSON. **Riak** is a key/value system design that delivers powerful yet simple data models for storing massive amounts of unstructured data and it is compatible with Amazon S3 and OpenStack Swift, easy integration, hide data Accuracy and availability. A zero-hop distributed hash table (ZHT), which has been tuned for the requirements of high-end computing systems.

Memcached is a high-performance, scalable, and distributed memory object caching system. There is no master node, all nodes are equal, there is no replication, and node selection is done by the client hashing algorithm. In our system design it just has an arbitrary set of servers, to each of which each client establishes a client/server connection. The Memcached client library is responsible for sending requests to the correct servers. The user application requests all keys using the client library, which calculates key hash values, determining which Memcached server should receive requests. The Memcached client sends parallel requests to all relevant Memcached servers. The Memcached servers send responses to the client library. The Memcached client library aggregates responses for the application. Each Memcached instance is totally independent, and does not communicate with others.

MongoDB uses sharding method for distributing data across multiple machines. In our MongoDB system design we have three following components in our sharded cluster: 1. Shard - Each shard contains a subset of the sharded data. Each shard can be deployed as a replica set. 2. Mongos (Query Routers) - The mongos acts as a query router, providing an interface between client applications and the sharded

cluster. 3. Config servers - Config servers store metadata and configuration settings for the cluster. As of MongoDB 3.4, config servers must be deployed as a replica set (CSRS).

Riak has a flexible key-value data model profile and session management, real-time big data, catalog, content management, customer 360, digital messaging and more use cases. Buckets are used to define a virtual keyspace for storing Riak objects. They enable you to define non-default configurations over that key space concerning replication properties and other parameters. Each host in the cluster runs a single instance of Riak, referred to as a Riak node. Each Riak node manages a set of virtual nodes, or vnodes, that are responsible for storing a separate portion of the keys stored in the cluster. Internally, Riak computes a 160-bit binary hash(20 bytes) of each bucket/key pair and maps this value to a position on an ordered ring of all such values. This ring is divided into partitions, with each Riak vnode responsible for one of these partitions (we say that each vnode claims that partition). The nodes of a Riak cluster each attempt to run a roughly equal number of vnodes at any given time. In the general case, this means that each node in the cluster is responsible for 1/(number of nodes) of the ring, or (number of partitions)/(number of nodes) vnodes. If two nodes define a 16-partition cluster, for example, then each node will run 8 vnodes. Nodes attempt to claim their partitions at intervals around the ring such that there is an even distribution amongst the member nodes and that no node is responsible for more than one replica of a key.

Comparison and Contrasts between our chosen 3 systems (Memcached, MongoDB, Riak) to ZHT:

Features	ZHT	Memcached	Mongo DB	Riak
Distributed key-value	Yes	Yes	Yes	Yes
Primary database Model	Key-Value Store	Key-Value Store	Document Store	Key-Value Store
Data Scheme	Schema free	Schema free	Schema free	Schema free
APIs and other access methods	No	Proprietary Protocol	Proprietary Protocol using JSON	HTTP API Native Erlang Interface
Replication Methods	Asynchronous replication	None	Master-Slave replication	Selectable replication factor
Consistency	Weak Consistency	None	Eventual Consistency	Eventual Consistency
Durability	Yes	Yes	Yes	Yes
Concurrency	Yes	Yes	Yes	Yes
Fault Tolerance	Yes	Yes	Yes	Yes

Evaluation Scale and Metrics:

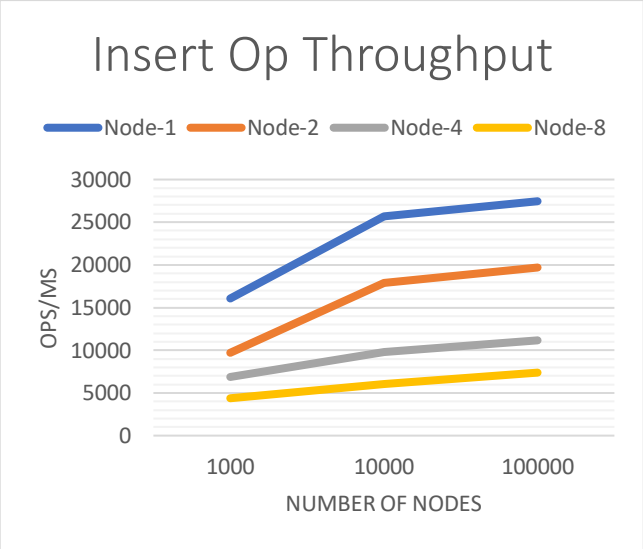
Memcached - Setup Memcached Commands:

MongoDB – Setup MongoDB Commands:

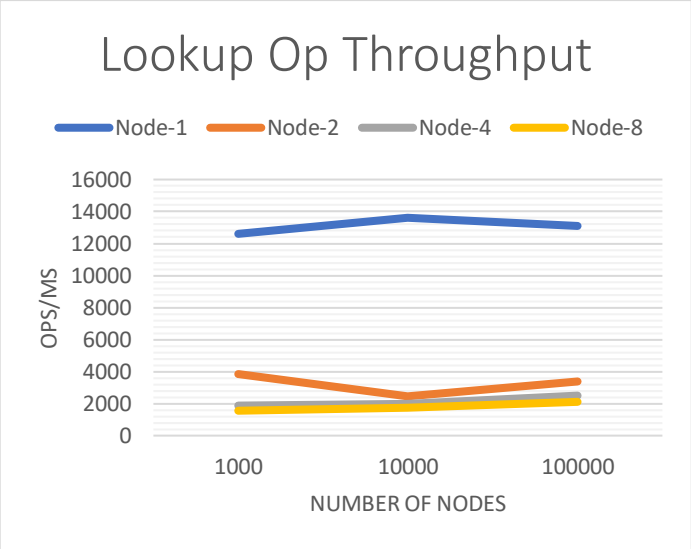
Riak – Setup Riak Commands:

Memcached – Throughput:

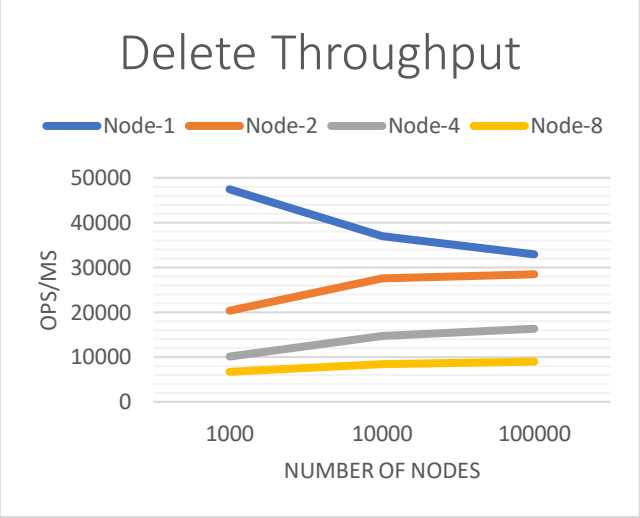
Insert throughput



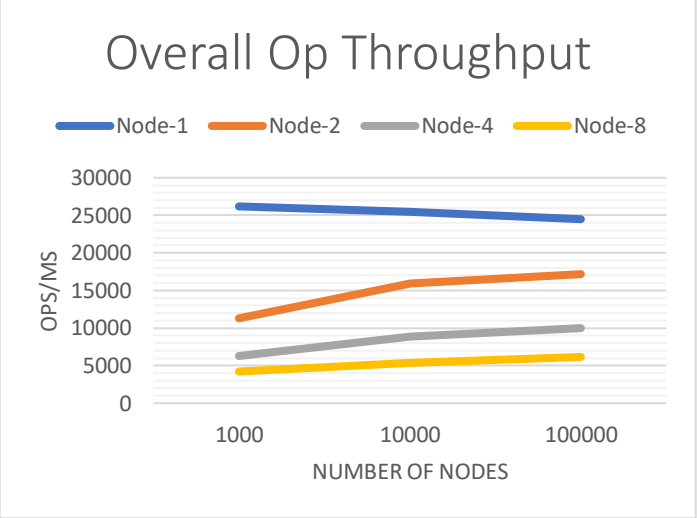
Lookup throughput



Delete Throughput

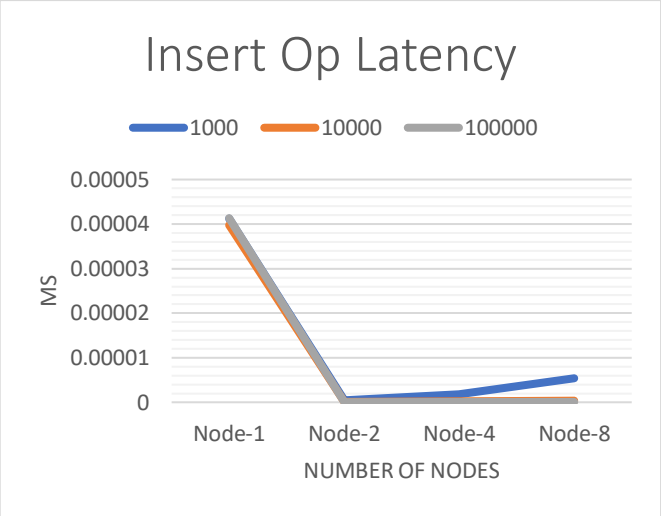


Overall Throughput

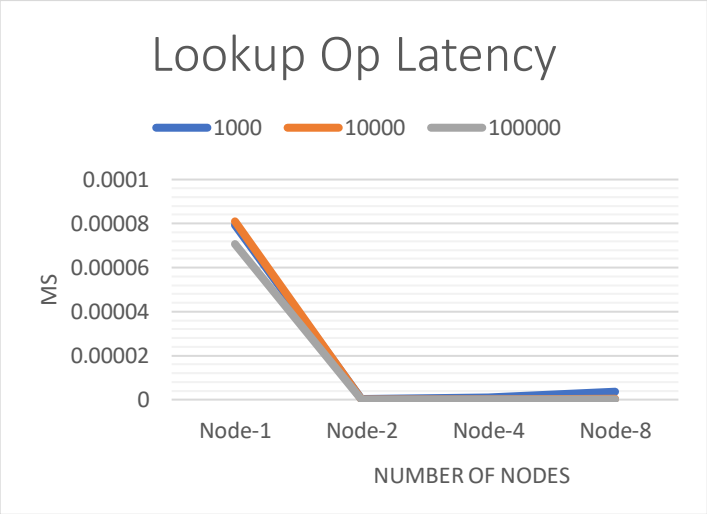


Memcached Latency:

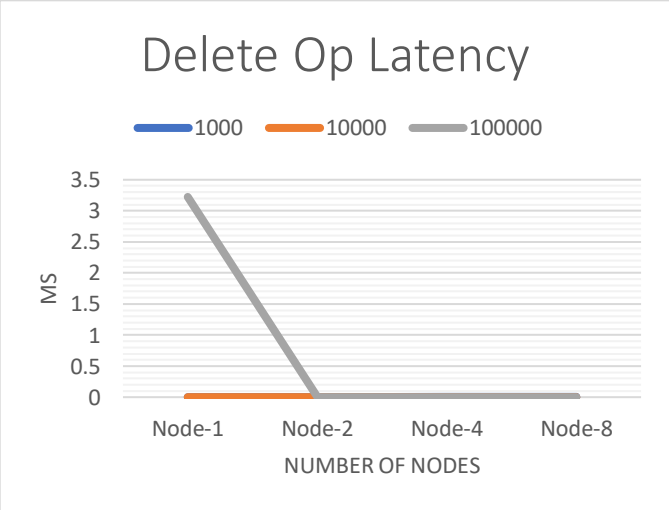
Insert Latency



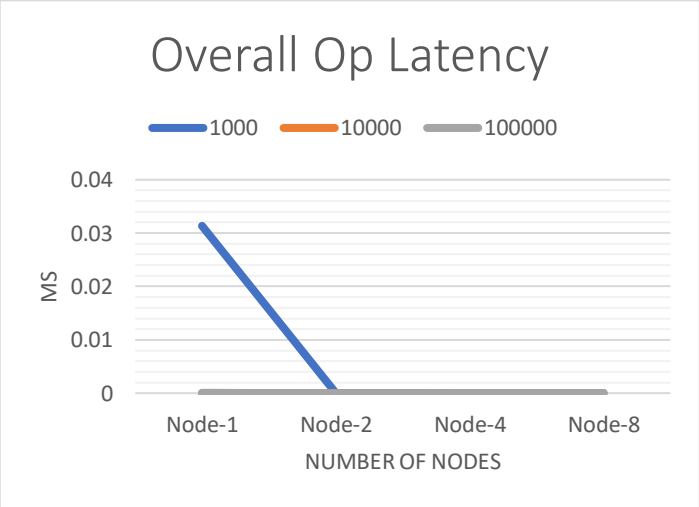
Lookup Latency



Delete Latency

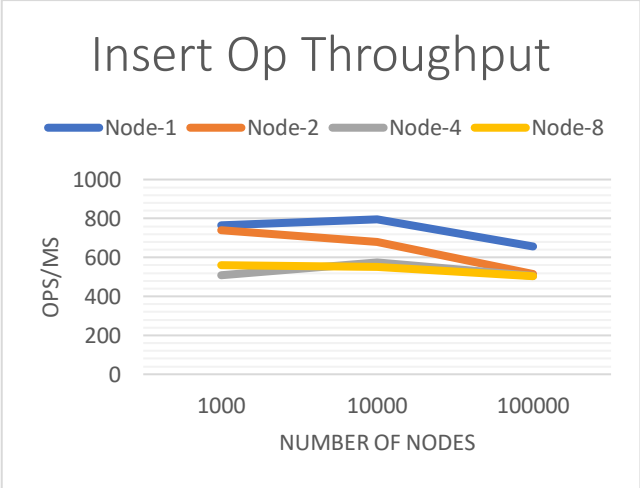


Overall Latency

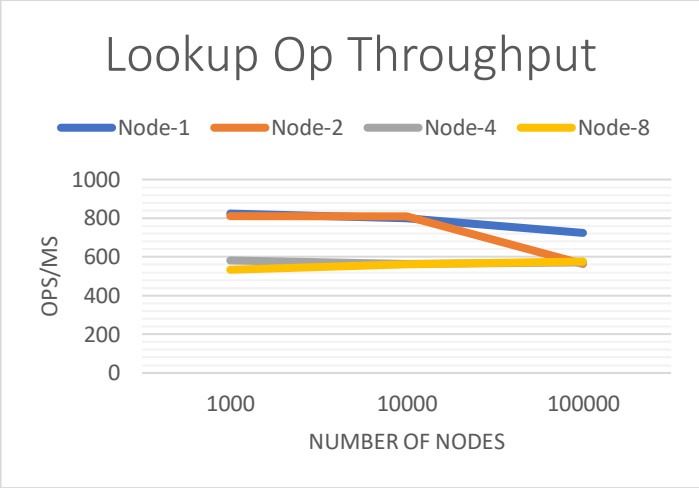


MongoDB – Throughput:

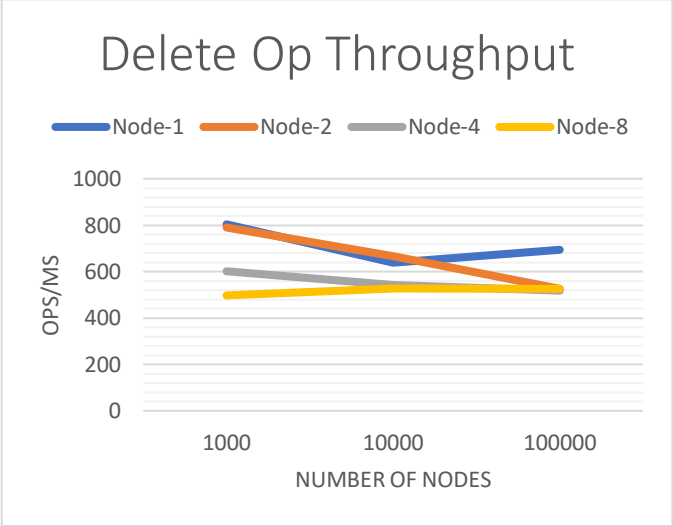
Insert Throughput



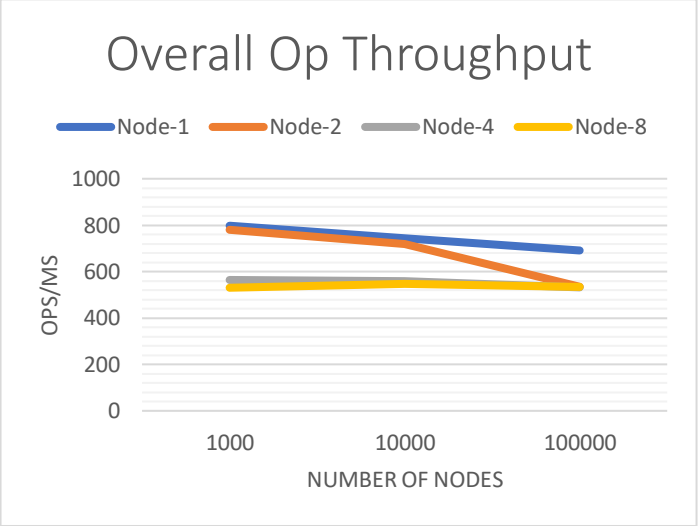
Lookup Throughput



Delete Throughput

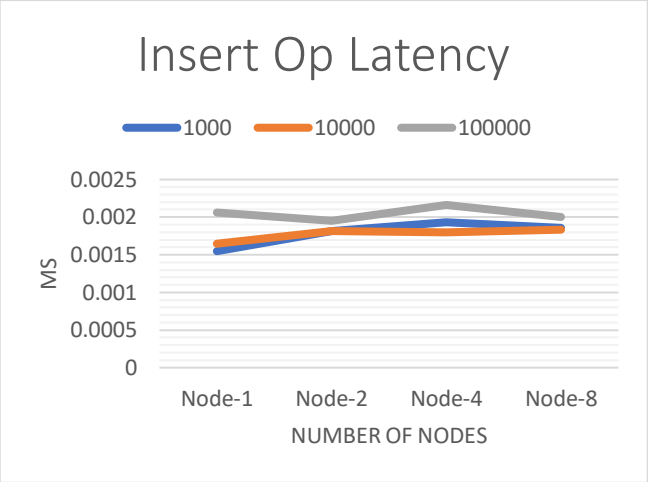


Overall Throughput

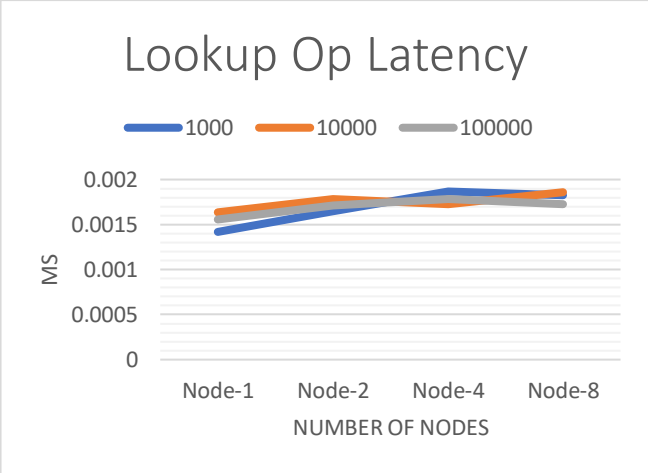


MongoDB – Latency:

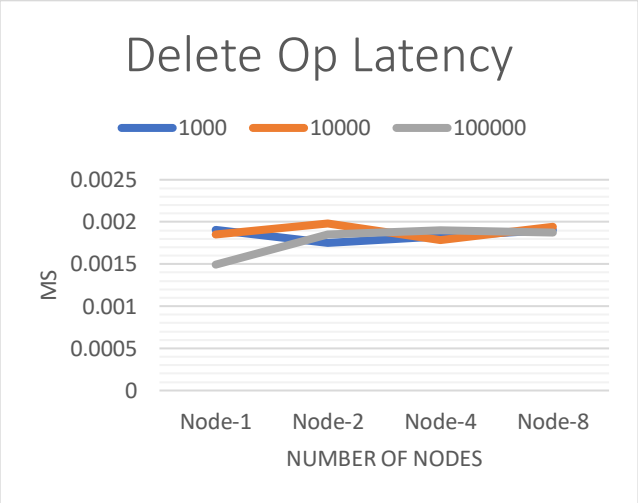
Insert Latency



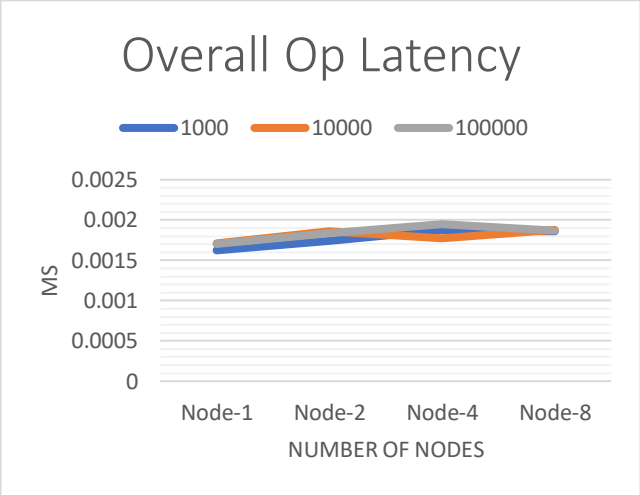
Lookup Latency



Delete Latency

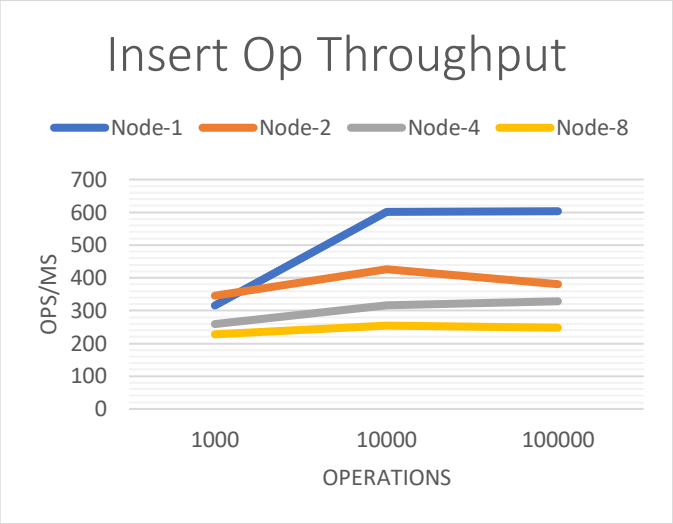


Overall Latency

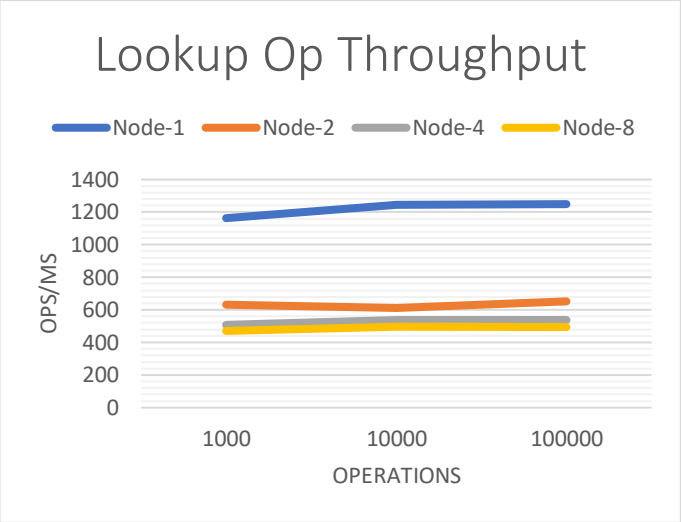


Riak – Throughput:

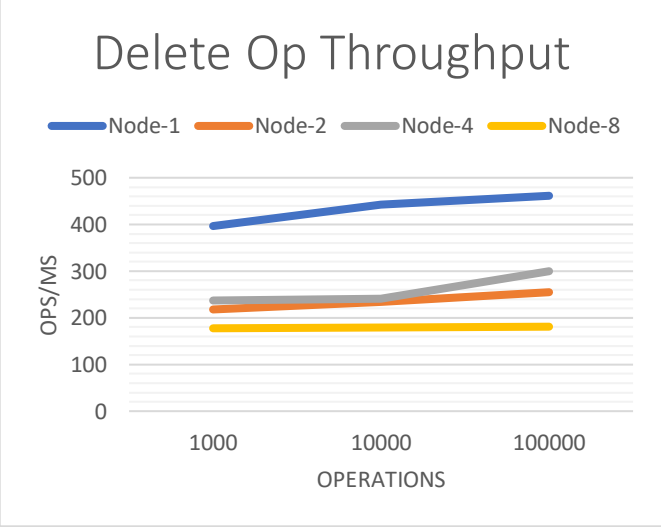
Insert Throughput



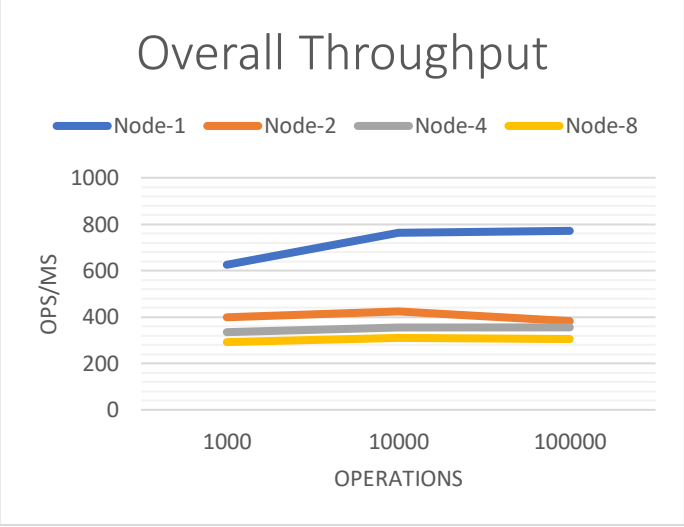
Lookup Throughput



Delete Throughput

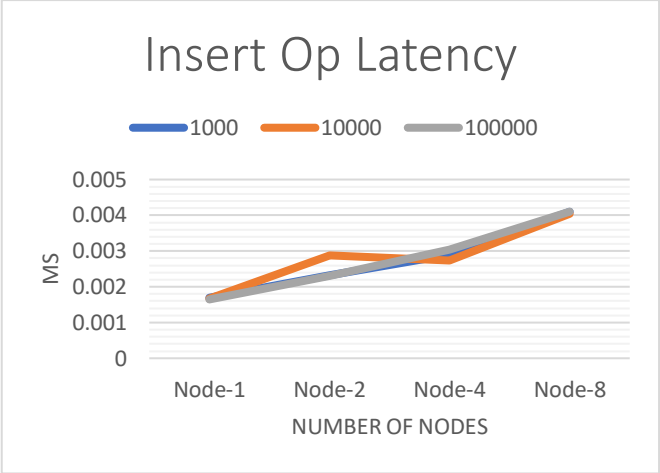


Overall Throughput

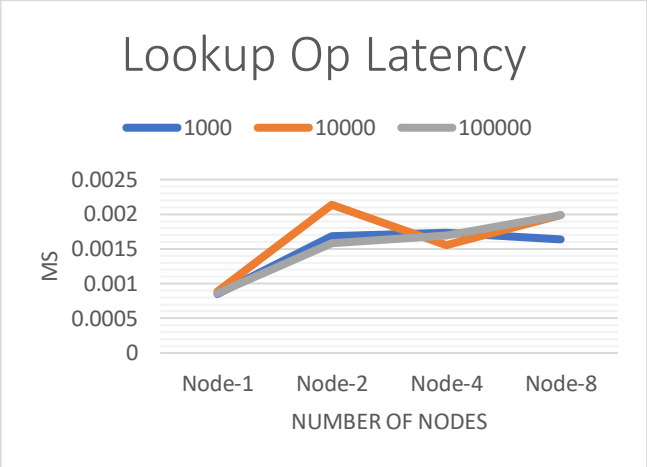


Riak – Latency:

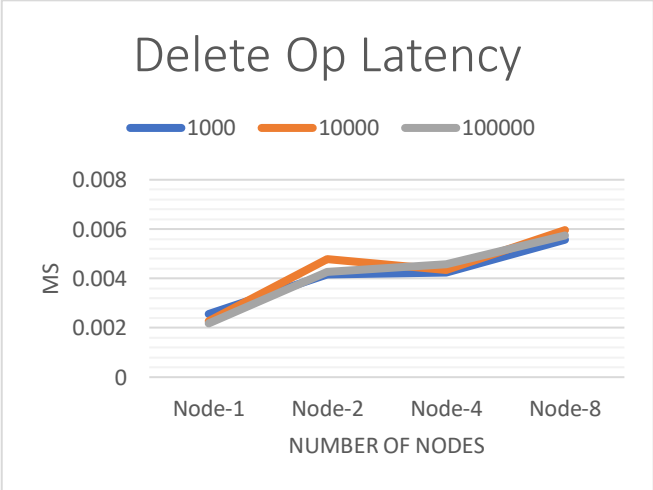
Insert Latency



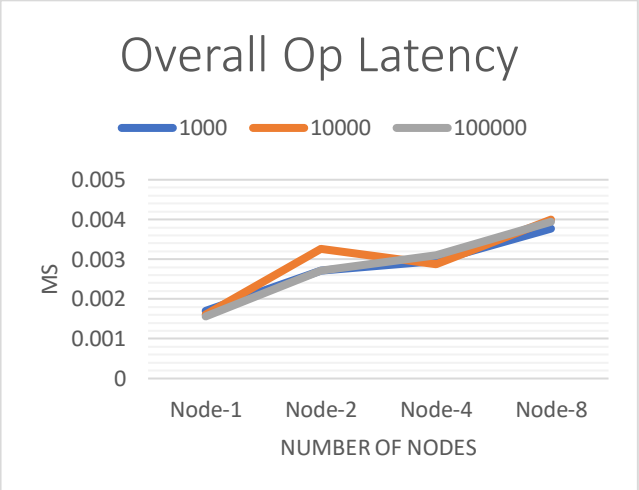
Lookup Latency



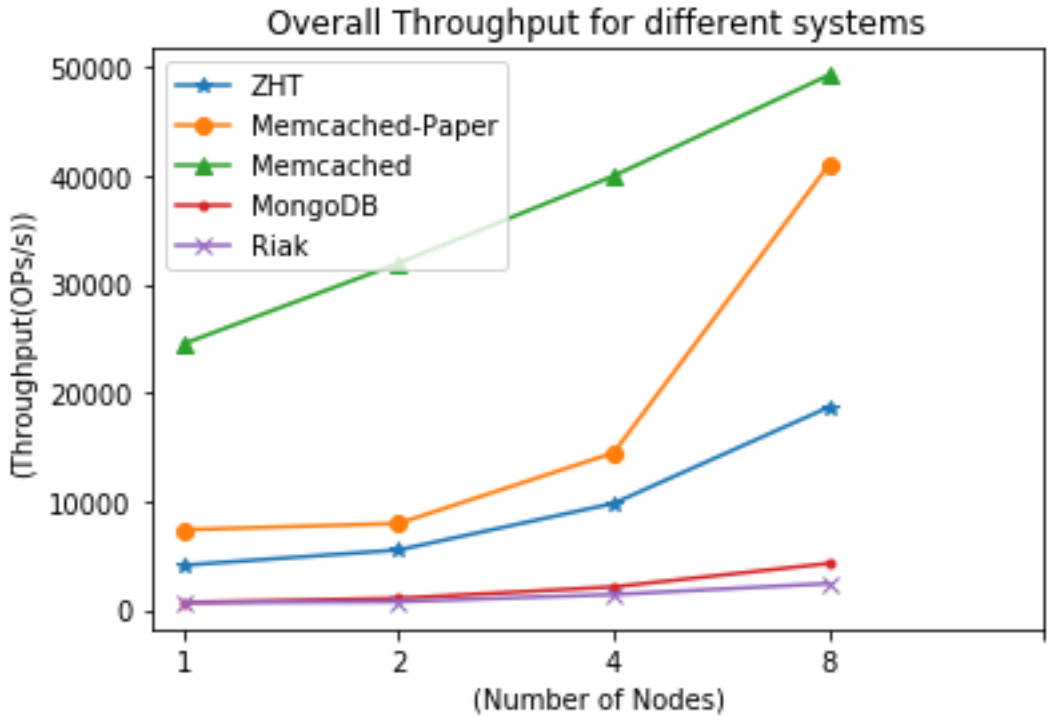
Delete Latency



Overall Latency

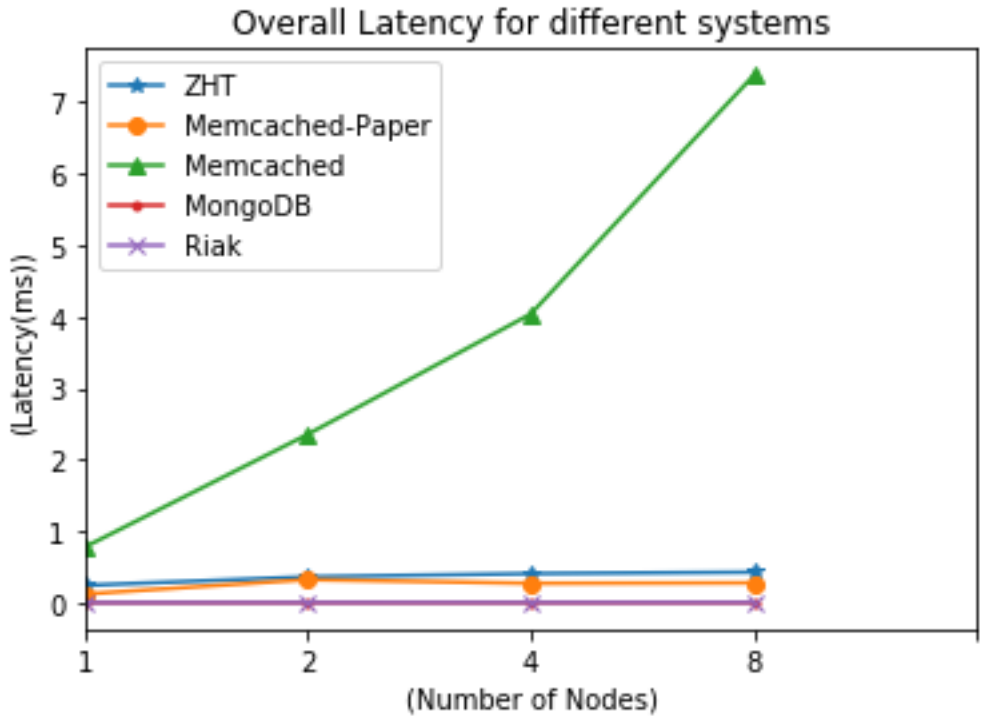


Overall Throughput comparison for our selected 3 systems and ZHT:



	1-Node	2-Node	4-Node	8-Node
ZHT	4117	5524	9813	18680
Memcached-Paper	7385	7961	14480	40995
Memcached	24486.7687	31919.0688	39960.8064	49237.8867
MongoDB	690.9691	1069.2928	2127.7476	4282.5387
Riak	671.2157	764.8504	1421.4025	2433.0373

Overall Latency comparison for our select 3 system and ZHT:



	1-Node	2-Node	4-Node	8-Node
ZHT	0.243	0.362	0.408	0.428
Memcached-Paper	0.122	0.324	0.272	0.278
Memcached	0.774	2.353	4.038	7.378
MongoDB	0.001704	0.001838	0.001865	0.001949
Riak	0.00156	0.00272	0.00309	0.00394

Conclusion:

In this project, we evaluated 3 distributed key-value storage system for different number of servers (1, 2, 4, 8). From our noted result, Memcached have maximum range of throughput so we can use these key-value storages for quick access of value compare to other. MongoDB and Riak performance remained consistent for all operations across multiple nodes, difference in performance can be verified on high-end computing systems.

References:

1. Yishan Li, Sathiamoorthy Manoharan, "A performance comparison of SQL and NoSQL databases", proc. of Pacific Rim Conference on Communications Computers and Signal Processing (PACRIM) IEEE, pp. 15-19, 2013.
2. Tan, Zhipeng & Dang, Yongxing & Sun, Jianliang & Zhou, Wei & Feng, Dan. (2014). PaxStore : A Distributed Key Value Storage System. 8707. 471-484. 10.1007/978-3-662-44917-2_39.
3. Zhong, Yunqin & Sun, Shangchun & Liao, Haojun & Zhao, Yanwei & Fang, Jinyun. (2011). A novel method to manage very large raster data on distributed key-value storage
4. http://pages.cs.wisc.edu/~vijayc/papers/memcached.pdf
5. https://docs.mongodb.com/manual/sharding/
6. Riak, 09 2011, [online] Available: http://basho.com/Riak.html