CS 553 HW7

An Empirical Evaluation of Distributed Key/Value Storage Systems

Pradyothan Govrineni
Bhavya Chawla

Files:
For cassandra
Insert.java
Delete.java
Lookup.java

For redis
RedisPut.c

For MongoDB
Putdata.c

Compare and contrast your 3 evaluated systems to ZHT:

1. Write a paragraph about each of your 3 systems, as well as about ZHT summarizing what the systems are, and what the key features are.

Cassandra is an accessible NoSQL data storage system run by the nonprofit Apache that uses a distributed design to provide high availability and dependability. When it comes to high availability, scalability, and dependability, Cassandra is described as an open-source NoSQL data storage system that relies on a distributed design. Open-source accessibility - Nothing is more thrilling than receiving a useful item for nothing. This is undoubtedly one of the key reasons for Cassandra's widespread acceptance and appeal.

Cassandra is designed to execute over numerous nodes rather than a single one. Without a master node, each node's relevance is equal, and there is no bottleneck that slows down the operation. Cassandra's scalability once again is due to the nodal architecture. Cassandra is intended to grow horizontally as your needs as a developer or company grow. Scaling-up in Cassandra is very easy and not limited to location. Adding or removing extra nodes can adjust your database system to suit your dynamic needs. Cassandra's capacity to replicate data is

largely what makes it fault-tolerant. Data replication refers to a system's capacity to store the same data across numerous nodes or locations. This increases its availability and system fault tolerance. As data has been duplicated and stored among other nodes in the cluster, the failure of a single node or data center does not put the system to a complete stop. Replication of data results in extensive backup and recovery. Because of Cassandra's configurable consistency feature, the developer can select one of the two kinds depending on the task at hand. The developer is free to employ either type of consistency or both at any time.

Remote Dictionary Server, or Redis, is a quick, free, in-memory key-value data storage. Redis has response rates that are less than one millisecond, enabling millions of queries per second for real-time applications in sectors including gaming, ad-tech, financial services, healthcare, and IoT. Redis has been rated the "Most Loved" database by Stack Overflow for five years running, making it one of the most well-liked open source engines in use today. Redis is a well-liked option for caching, session management, gaming, leaderboards, real-time analytics, geospatial, ride-hailing, chat/messaging, video streaming, and pub/sub apps because of its quick speed.

For storing large amounts of data, **MongoDB** is a document-oriented NoSQL database. MongoDB uses collections and documents rather than the tables and rows seen in conventional relational databases. The fundamental unit of data in MongoDB is a pair of key-value pairs, which make up documents. Collections are the equivalent of relational database tables in that they store collections of documents and perform certain functions. A database called MongoDB first became popular in the middle of the 2000s.

ZHT has been adjusted to meet the needs of powerful computer systems. Future distributed systems including parallel and distributed file systems, distributed job management systems, and parallel programming systems are all intended to be built on top of ZHT. At extremely large sizes of millions of nodes, ZHT aims to achieve high availability, robust fault tolerance, high throughput, and low latencies. ZHT possesses some significant characteristics, including being lightweight, dynamically allowing nodes to join and leave, persistent, scalable, fault-tolerant through replication, and supporting unusual operations like append (providing lock-free concurrent key/value modifications in addition to insert/lookup/remove).

2. Follow this writeup with a comparison that discusses the similarities and differences between your 3 systems to ZHT.

From the conclusion it is said that the similarities are that all systems consist of key pairs and contain no sql databases. All systems contain scalable and accessible systems that allow easy access to use and they are open source to code from.

The theoretically distinct data storage models of MongoDB and Redis are the most visible distinction between them. While MongoDB's default storage engine, WiredTiger, saves data on disk, Redis is an in-memory data store. A horizontal scale-out design has always been a part of MongoDB. Sharding, which enables you to distribute data across numerous nodes, achieves horizontal scaling. Using the specified shard key as a basis, data is divided among nodes. Redis

Cluster, which shares data across numerous nodes, is Redis's answer to scaling out. MongoDB stores data in JSON document format, whereas Cassandra stores data in Column-Family structure. Secondary indexes are not advised in Cassandra since they reduce performance. For greater speed and to prevent having to look through every document in MongoDB, indexes are suggested.

Cassandra is a fantastic option for high write throughput, however MongoDB is a better alternative if your application requires extremely high read concurrency. MongoDB has a single master, but Cassandra has no master nodes and all nodes are peers. Cassandra finally duplicates the written data to the number of cluster nodes and cluster nodes in various geographic locations that are defined in the replication factor. Replication in MongoDB takes some preparation. If the primary database fails, you can set up a secondary database that will be automatically selected. In MongoDB, reads are initially replicated to secondary replicas after which they are committed to the primary replica. The consistency and ability are similar in all databases.

ZHT is developed and built to serve as a basis for the creation of fault-tolerant, high-performance, and scalable storage systems. It is optimized for high-end computing systems, in this way it is slightly comparable to REDIS. ZHT may maintain low overheads while enduring multiple failures. Additionally adaptable, it supports the joining and leaving of dynamic nodes. ZHT has the potential to be an effective distributed key-value store and an essential component of large-scale distributed systems like file systems and task schedulers. In this way ZHT has overall more power than the other noSQL databases.

3. Create a table of features that clearly shows how these systems are different. Evaluate these systems not just on performance, but scalability, and ease of use.

Features	Cassandra	MongoDB	Redis
Decentralization	Unlike other systems, Cassandra does not have a single point of failure. Each node handles the requests in the same way	It has master and slave approach	It also has Master and Slave approach. If master node is lost, can temporarily make slave as a master node
Scalability	You may easily add more hardware to satisfy those needs as data and demands grow. You don't need to shut down the database or make any significant	Horizontal scaling is offered by MongoDB in terms of scalability. You may spread large datasets across numerous servers in this fashion.	A Redis database may be readily scaled without any downtime or performance loss.

	changes because of its horizontal scalability.		
Ease of use	Cassandra provides CQL (Cassandra Query Language) as an alternative to SQL. Connect to Cassandra using this straightforward interface.	The simplicity of usage is one of the finest aspects of MongoDB. Focusing on documents, MongoDB. It uses the document notion to store data, which is more flexible than traditional databases.	Redis is not only simple to use, but it also makes code simpler by cutting down on the amount of lines needed to store and retrieve it's data.
Flexibilty	You can store organized, semi-structured, and unstructured data with Cassandra. If desired, Cassandra may be configured to use several data centers. The distribution of data is facilitated by this.	Schemas are absent in MongoDB. Any form of data can be stored in separate documents.	Redis permits key-value pairs to be stored in files up to 512 MB in size..
Performance	Cassandra can handle a large number of write requests rapidly and effectively without compromising the read requests.	One of MongoDB's key benefits is the ability to index documents, which enables quicker document access.	Redis keeps a large amount of data in memory, and it responds to transactions significantly more quickly than Cassandra, which persists data to disk via conventional read-write transactions.
Sharding	Cassandra supports sharding. Data can be distributed between nodes using a method akin to key-based sharding, but automatically.	Sharding is an efficient method for dividing data into smaller portions, and MongoDB supports it. This method may be used to store data sets of any size. They can also be spread	Redis too support Sharding, may automatically spread your data over several nodes using Redis Cluster, a native sharding implementation built into Redis that

		over several servers.	eliminates the need for third-party tools and services.

Evaluation Scale and Metrics

Dataset taken for the given databases:

Cassandra: 0.05MB
MongoDB: (20MB)
Redis: (200MB)

Latency (in msec):

Insert:

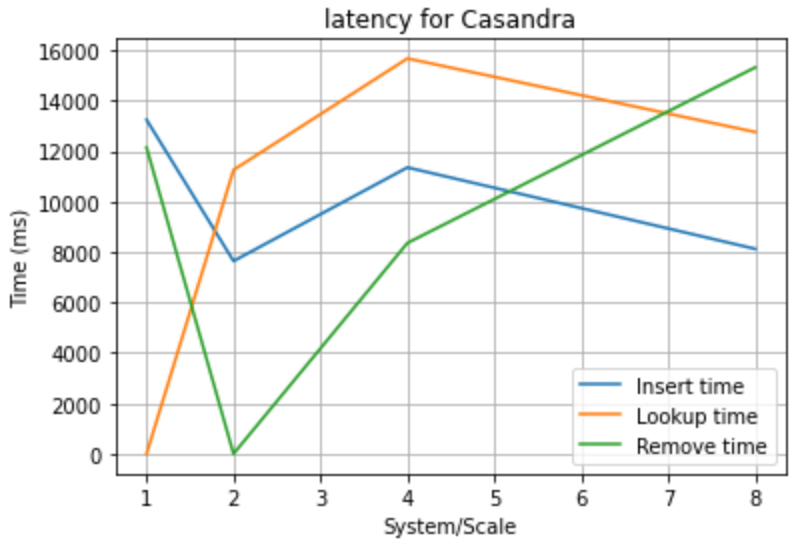
System / Scale	1	2	4	8
Cassandra	13248	7647	11358	8128
MongoDB	0.061	0.6357	0.5398	-
Redis	50538	114278.5	144328.2	154932.52

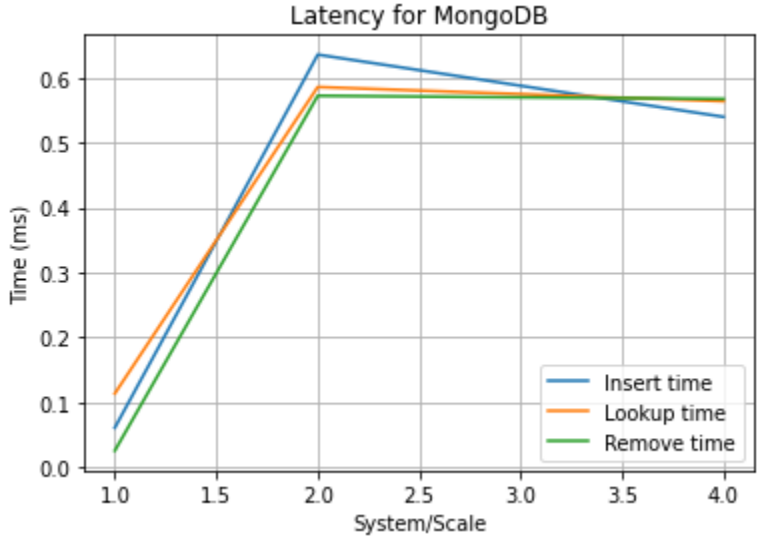
Lookup:

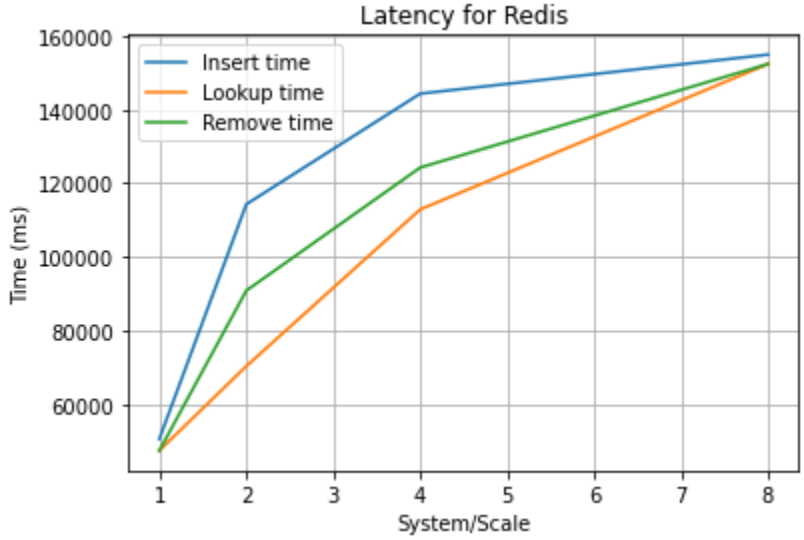
System / Scale	1	2	4	8
Cassandra	3	11260	15680	12764
MongoDB	0.1137	0.5858	0.5639	-
Redis	47390	70293.6	112876.75	152385.46

Remove:

System / Scale	1	2	4	8
Cassandra	12149	0.8	8364	15326.86
MongoDB	0.0257	0.5723	0.5675	NA
Redis	47363	90823	124254	152376.5


latency for Casandra
Time (ms)
System/Scale
Insert time
Lookup time
Remove time


Latency for MongoDB
Time (ms)
System/Scale
Insert time
Lookup time
Remove time


Latency for Redis
Time (ms)
System/Scale
Insert time
Lookup time
Remove time

Throughput (MB/sec): (dataset (MB) / Latency(sec)):

Dataset taken for the given databases:

Cassandra: (0.05MB)
MongoDB: (20MB)
Redis: (200MB)

Calculating throughput with dataset divided by the time(latency in sec))

For example:
Insert (Mb/sec): Insert ops in cassandra with 1 node, (0.05/13.248)=264.96 MB/s

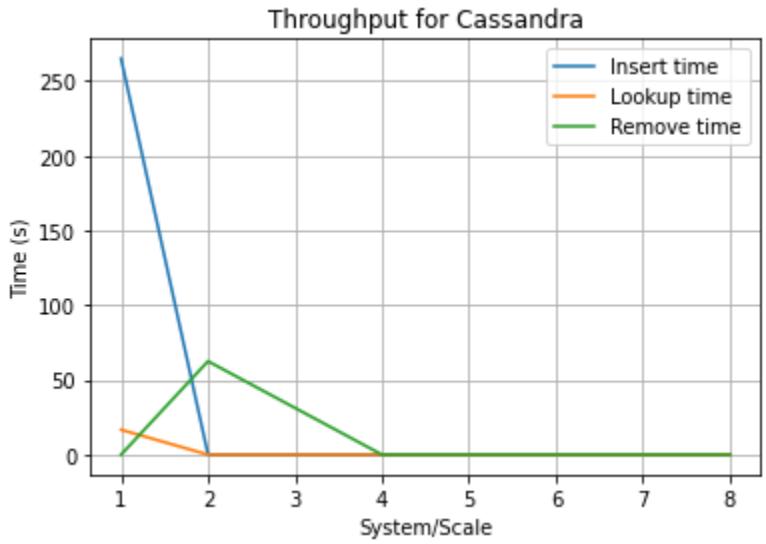
System / Scale	1	2	4	8
Cassandra	264.96	0.006538511835	0.004402183483	0.006151574803
MongoDB	0.00000305	31461.38115	37050.75954	
Redis	0.25269	1.750110476	1.385730578	1.290884573

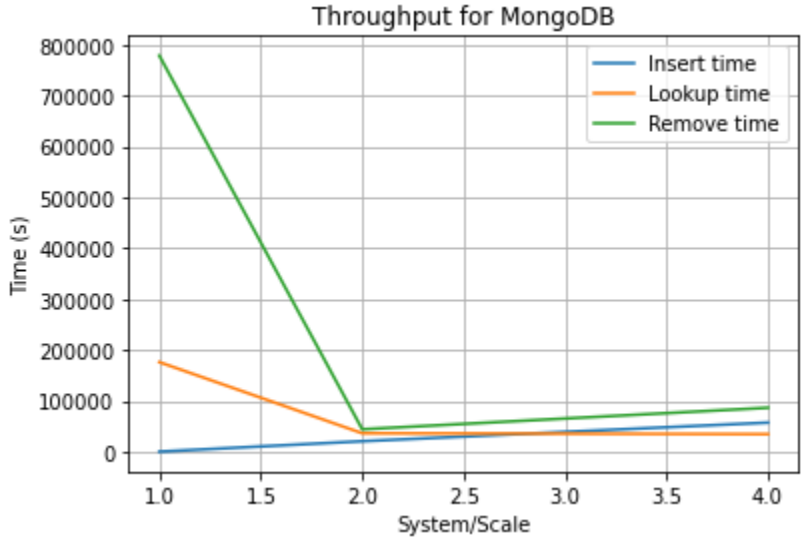
Lookup (Mb/sec):

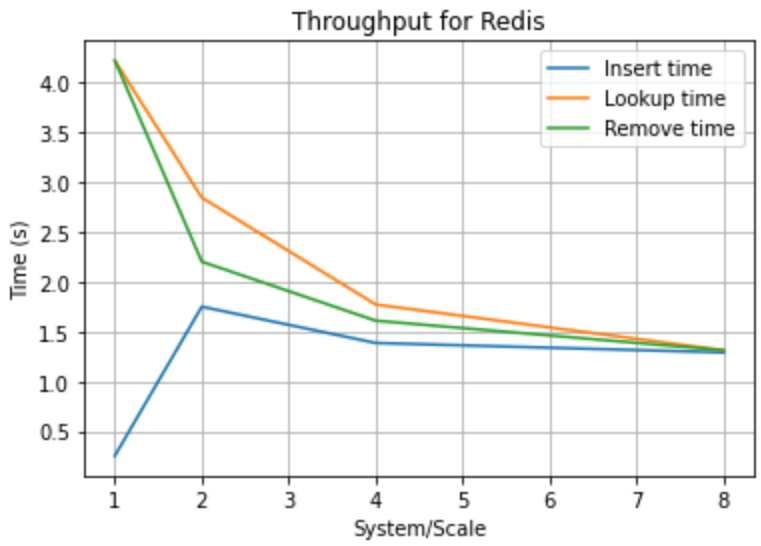
System / Scale	1	2	4	8
Cassandra	16.66666667	0.004440497336	0.003188775514	0.003917267314
MongoDB	175901.4952	34141.34517	35467.28143	
Redis	4.220299641	2.845209237	1.771844069	1.312461176

Remove(Mb/sec):

System / Scale	1	2	4	8
Cassandra	0.004115565067	62.5	0.005978000956	0.003262246801
MongoDB	778210.1167	34946.70627	35242.29075	
Redis	4.222705487	2.202085375	1.609606129	1.312538351


Throughput for Cassandra
Insert time
Lookup time
Remove time
Time (s)
System/Scale
0
50
100
150
200
250
1
2
3
4
5
6
7
8


Throughput for MongoDB
Insert time
Lookup time
Remove time
Time (s)
System/Scale
0
100000
200000
300000
400000
500000
600000
700000
800000
1.0
1.5
2.0
2.5
3.0
3.5
4.0


Throughput for Redis
Insert time
Lookup time
Remove time
Time (s)
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
1
2
3
4
5
6
7
8
System/Scale