Rakesh Abbireddy

Cassandra

An open-source distributed database system called Cassandra is made to manage massive volumes of data across several commodity computers. High scalability and fault tolerance are key architectural features of the distributed database Cassandra. With several server nodes, it can manage massive volumes of data without experiencing a single point of failure.. It is resistant to system faults and unique in that it may grow linearly while preserving high availability. Cassandra offers customers configurable consistency—that is, it may be configured for eventual consistency or strong consistency—based on their needs because it does not have a single point of failure. Its column-oriented, adaptable data model is appropriate in situations where a relational database model might not be practical. Cassandra may be accessed by users who are accustomed to traditional database systems since it offers a query language that is comparable to SQL.

Key Features:

- Distributed architecture with no single point of failure.
- Scalable to handle large data sets spanning multiple data centers.
- Offers tunable consistency, allowing a trade-off between consistency and availability.
- Uses a distributed hash table (DHT) for data distribution.
- Combines features from Google's Big Table and Amazon's Dynamo.

MongoDB:

MongoDB is a NoSQL database that stores data in flexible, JSON-like documents. This makes it a good choice for storing and managing unstructured data, such as user profiles, product catalogs, and social media posts.

Key features:

- Stores data in flexible, JSON-like documents, enabling easy organization and querying of complex data structures.
- Embraces a schema-less approach, allowing for agile development and adapting to evolving data structures without rigid constraints.
- Designed for high performance and scalability, effectively handling large volumes of data and increasing workloads seamlessly.
- Ensures data availability and disaster recovery by creating multiple copies of data across multiple servers, providing redundancy and fault tolerance.

- scales data storage and retrieval by distributing data across multiple servers, improving performance and scalability as data grows.

ZHT

ZHT is a distributed key-value store designed for high-performance computing. It aims to provide a scalable, fault-tolerant, and high-performance environment for storing large volumes of data.

Key Features:

- Zero-hop architecture for efficient data access.
- Designed for use in high-performance computing environments.
- Provides high throughput and low latency.
- Supports data replication and fault tolerance.
- Efficient in handling small metadata

Comparison of Cassandra, mongoDB, and ZHT:

Each of the three systems is built to support massive data volumes and scale horizontally. Distributed systems like Cassandra, MongoDB, and ZHT are made to operate over several nodes. Although they do this through various methods (distribution of data in ZHT and replication in Cassandra and MongoDB), they provide significant degrees of fault tolerance

MongoDB is document-oriented, ZHT is a key-value store, while Cassandra employs a wide-column store. These systems' approaches to modeling, storing, and querying data are impacted by this basic distinction. While MongoDB offers excellent consistency among shards, Cassandra offers customizable consistency, while ZHT usually concentrates on ultimate consistency. Cassandra contains a query language similar to SQL, ZHT enables simple key-value operations, and MongoDB has sophisticated query capabilities. Cassandra offers lightweight transactions, ZHT lacks native transaction capability, and MongoDB allows multi-document ACID transactions.

Feature	Cassandra	MongoDB	ZHT
Data Model	Wide-column store	Document-oriented	Key-Value store
Primary Use Case	High write throughput, scalability	Flexible data model, ad-hoc queries	High-performance, scalable key-value storage
Query Language	CQL (Cassandra Query Language)	MQL (MongoDB Query Language)	API-based (C++, Python, etc.)
Consistency	Tunable consistency levels	Strong consistency within a shard	Eventual consistency
Partitioning	Automatic sharding based on partition keys	Sharding based on user-defined shard keys	Hash-based partitioning
Replication	Masterless, multi-master replication	Master-slave replication	Replication support varies
Transactions	Limited support (Lightweight transactions)	ACID transactions (since v4.0)	No native transaction support
Scalability	Linearly scalable	Horizontally scalable	Designed for high scalability
Fault Tolerance	High (due to replication)	High (replica sets)	Depends on implementation

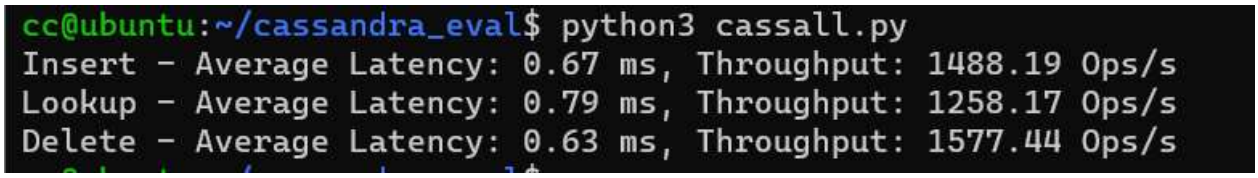
Data Distribution	Even distribution using consistent hashing	Data distribution across shards	Uniform data distribution
Read/Write Performance	High write performance	Balanced read/write performance	Optimized for fast data retrieval
Indexing	Secondary indexing support	Comprehensive indexing options	Limited indexing capabilities
Community & Support	Large community, extensive documentation	Very popular, extensive support	Smaller community, limited documentation
License	Open-source (Apache License)	Open-source (SSPL) and commercial versions	Open-source

Cassandra analysis

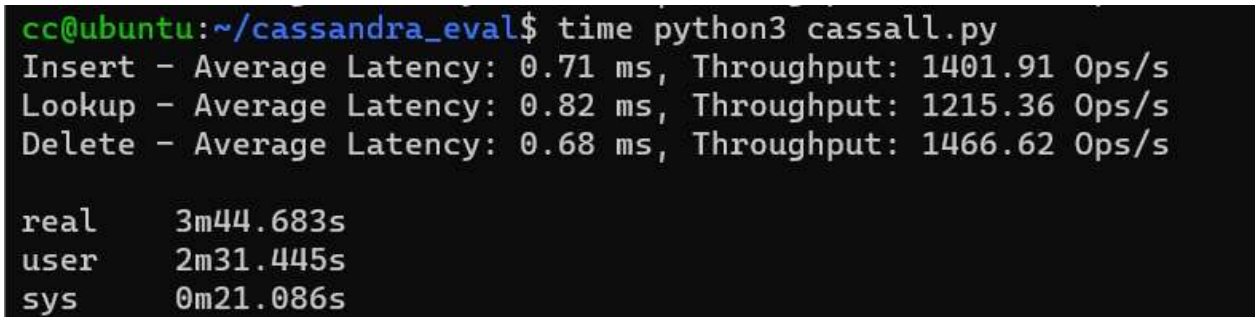
Cassandra all to all communication

- I found out that 1000000 records are good for stress testing the systems

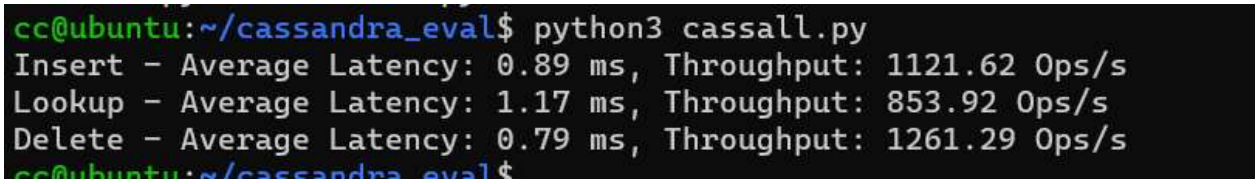
1 node with 1000000 recodes



2 nodes with 100000 records



4 nodes with 100000 records



8 nodes 100000 records

```
Insert - Average Latency: 1.03 ms, Throughput: 974.04 Ops/s
Lookup - Average Latency: 1.25 ms, Throughput: 800.15 Ops/s
Delete - Average Latency: 0.82 ms, Throughput: 1220.11 Ops/s
```

All-to-all communication is achieved for load balancing

```
CONTAINER ID   NAME              CPU %     MEM USAGE / LIMIT   MEM %     NET I/O           BLOCK I/O         PIDS
4907f0e90d74   compose_node2_1   45.05%    8.354GiB / 12GiB    69.62%    40.2MB / 37.6MB   8.19kB / 17.4MB   140
62d7afe62ae8   compose_node3_1   44.32%    8.355GiB / 12GiB    69.62%    38.1MB / 35.8MB   8.19kB / 16.5MB   139
14502d5ca3c5   compose_node1_1   44.84%    8.36GiB / 12GiB     69.66%    39.2MB / 37.1MB   0B / 16.9MB       156
3fef7763946e   compose_node6_1   42.73%    8.358GiB / 12GiB    69.65%    39.4MB / 37.1MB   4.1kB / 23.5MB    144
457bde57927c   compose_node8_1   44.00%    8.353GiB / 12GiB    69.61%    40.8MB / 38.7MB   4.1kB / 19.8MB    142
d14a543cda06   compose_node5_1   44.37%    8.353GiB / 12GiB    69.61%    39.6MB / 39.1MB   4.1kB / 19.1MB    138
dea821d19d63   compose_node7_1   44.45%    8.351GiB / 12GiB    69.60%    39.4MB / 36.7MB   12.3kB / 19.7MB   141
a4c3b05f5233   compose_node4_1   43.64%    8.382GiB / 12GiB    69.85%    411MB / 396MB     24.6kB / 166MB    162
```

Amount of CPU and RAM usage for 8 nodes for Cassandra nodes displayed.

Number of Nodes	Insert (ms)	Lookup (ms)	Delete (ms)
1 Node	0.67	0.79	0.63
2 Nodes	0.71	0.82	0.68
4 Nodes	0.89	1.17	0.79
8 Nodes	1.03	1.25	0.82

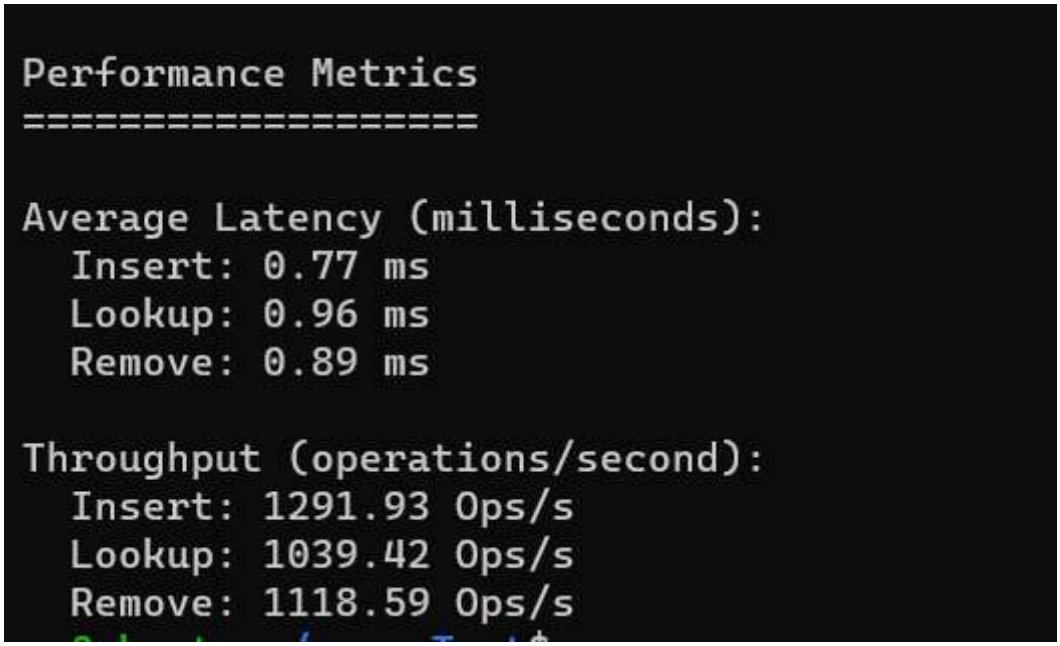
Number of Nodes	Average Latency (ms)
1 Node	0.70 ms
2 Nodes	0.74 ms
4 Nodes	0.95 ms

8 Nodes	1.03 ms

The performance analysis of Cassandra with different node configurations reveals a typical trade-off encountered in distributed systems: as the number of nodes increases, so does the average latency for database operations. Initially, in a single-node setup, the latency is the lowest at 0.70 milliseconds, reflecting the absence of network overhead and inter-node communication. However, as the nodes scale to 2, 4, and eventually 8 nodes, the average latencies gradually increase to 0.74 ms, 0.95 ms, and 1.03 ms, respectively. This increment can be attributed to the complexities introduced by distributed data storage, including network latency, data replication, synchronization costs, and potential bottlenecks in inter-node communication. While the increased latency signifies the overhead of maintaining a distributed system, it is balanced by the advantages such configurations provide, particularly in terms of fault tolerance, data redundancy, scalability, and load balancing. This scenario highlights the fundamental considerations in scaling distributed databases, where optimizing and balancing between latency, availability, and throughput becomes crucial, especially in high-availability and high-throughput scenarios.

MongoDB analysis

Nodes 2 with 100000 records



Similarly for all other nodes average is represented in the table

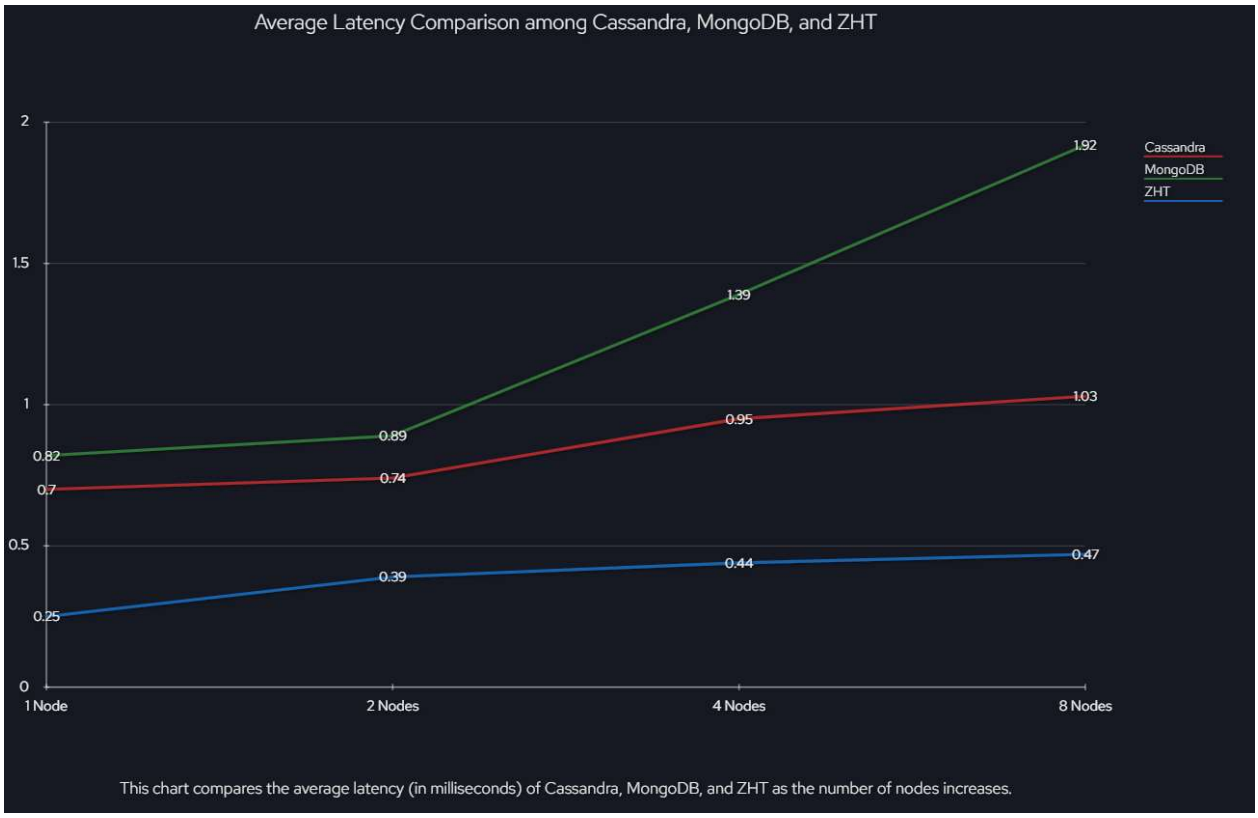
Number of Nodes	Average Latency (ms)
1 Node	0.82 ms
2 Nodes	0.89 ms
4 Nodes	1.39 ms
8 Nodes	1.92 ms

A frequent phenomenon in distributed database systems is vividly illustrated by the performance statistics for MongoDB across a range of cluster sizes, from 1 to 8 nodes: the average latency for operations increases as the cluster expands. The latency is lowest in a single-node configuration (0.82 ms), mostly because there are no network-related delays or replication costs. The delay does, however, marginally increase to 0.89 milliseconds when the cluster grows to two nodes, suggesting the initial effects of dispersed processes and data replication. In the four-node arrangement, this tendency is more noticeable as latency increases to 1.39 milliseconds. This is probably because of more network traffic, intricate data replication patterns,

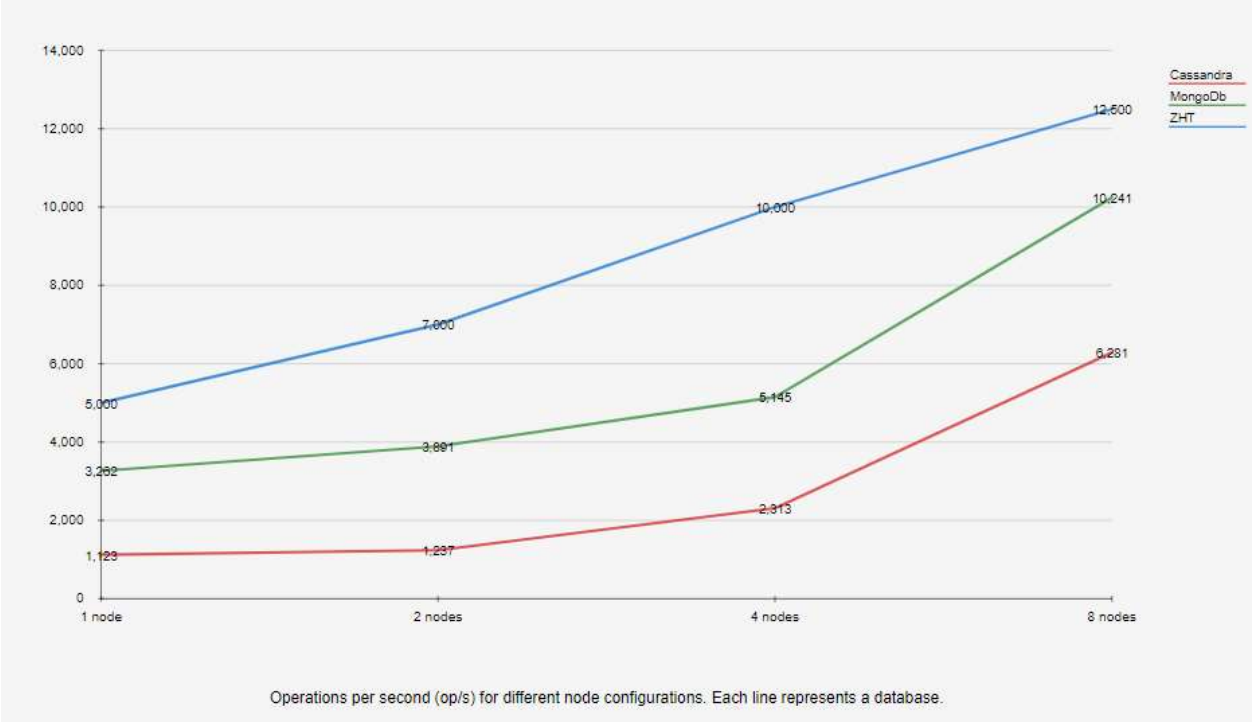
and node coordination overhead. In an eight-node configuration, the latency peaks at 1.92 milliseconds, highlighting the difficulties in maintaining a larger distributed system due to sophisticated inter-node communication and possible load-balancing issues.

The trade-offs involved in growing distributed databases such as MongoDB are highlighted by these observations. Because of the difficulties involved with distributed computing, increasing the number of nodes improves fault tolerance, data redundancy, and overall system resilience, but it also adds to delay. It is important to balance scalability, availability, and latency while designing and optimizing distributed systems. Utilizing the advantages of a distributed database system while reducing latency requires efficient cluster administration, well-thought-out data distribution plans, and frequent performance tweaking.

Overall analysis



Latency



Throughput

Latency: This is the time it takes for a system to respond to a request. In the context of databases, lower latency means that the database is able to handle requests more quickly, which is generally preferred, especially for real-time applications.

ZHT

Appears to have the best performance regarding latency. It's not only the lowest at the start but also shows minimal increase as more nodes are added, indicating that ZHT's performance is less affected by scale. This suggests that ZHT is quite efficient in handling distributed operations and maintaining quick response times even as the system grows.

MongoDB

Starts with moderate latency and gradually increases as more nodes are added. MongoDB's performance seems to degrade at a steady pace, which may indicate that while it can handle increased load, the response time does get slower. This could be due to various factors such as the overhead of maintaining consistency and synchronization across nodes.

Cassandra

Begins with a slightly higher latency than MongoDB and scales to a much higher latency as more nodes are added. This could suggest that Cassandra, while designed to be highly available and fault-tolerant, may incur more overhead as it scales. This overhead could be related to its consistency model or replication strategies, which might become more complex with more nodes.

Scalability

This refers to the ability of a system to maintain or improve performance as it is given more resources (like nodes). The graph shows ZHT as the most scalable in terms of latency performance, followed by MongoDB, with Cassandra being the least scalable of the three.

Base Throughput Comparison

ZHT

With a base throughput of 5000 op/s on a single node, ZHT shows superior initial performance. This suggests it's highly efficient or optimized for operations per second at the outset.

MongoDB

At 3262 op/s for a single node, MongoDB is significantly ahead of Cassandra but lags behind ZHT. This indicates MongoDB is well-optimized for single-node operations, offering a balanced mix of efficiency and speed.

Cassandra

: Starting at 1123 op/s, Cassandra shows the lowest initial throughput. This could imply that Cassandra might prioritize other aspects like data consistency or partition tolerance over raw speed in single-node configurations.

Scaling Efficiency

Cassandra

The scaling efficiency of Cassandra is nonlinear and most notable when moving from 4 to 8 nodes (from 2313 to 6281 op/s). This suggests that Cassandra's architecture benefits substantially from larger cluster sizes, possibly due to more effective data distribution and replication strategies.

MongoDB

MongoDB demonstrates a more linear scaling pattern. The increase from 3262 op/s (1 node) to 10241 op/s (8 nodes) suggests a consistent improvement in throughput with each node added. This linear pattern can be advantageous for predictable scaling.

ZHT

ZHT starts strong but shows diminishing returns as more nodes are added (scaling from 5000 op/s at 1 node to 12500 op/s at 8 nodes). This suggests that while ZHT is highly efficient in smaller configurations, its scaling efficiency decreases as the cluster size grows.

Base Throughput: ZHT > MongoDB > Cassandra