International Journal of Computer Trends and Technology
ISSN: 2231–2803 / https://doi.org/10.14445/22312803/IJCTT-V71I9P103

Volume 71 Issue 9, 15-21, September 2023

Original Article

Important Considerations for Maximizing Performance and Ensuring Uninterrupted Operation with the Cassandra Database

Venugopal Thati

VP IT Enterprise Architect, FL, USA.

Corresponding Author : vthati86@gmail.com

Received: 11 July 2023 Revised: 23 August 2023 Accepted: 06 September 2023 Published: 27 September 2023

Abstract *- In the world of digital transformation and applications made available globally by enterprises for their customers, it is becoming very important to keep systems online 24/7, incurring no downtime. Moreover, these systems must scale as the demand grows and perform optimally to meet business needs. One key component in any enterprise application is the database, and it is a challenging component to scale easily compared to the front-end web apps or backend services where no state is maintained. There has been tremendous growth over the last decade in distributed NoSQL databases, which promise to solve availability, scalability, and reliability challenges. Moreover, several top-tier companies successfully implemented solutions for their businesses using these distributed databases, and some of these companies are contributing to introducing more features in these databases. This article covers important facets of achieving zero unscheduled downtime, fault tolerance, high availability, and scalability with Cassandra. Cassandra can be self-managed or used as a managed service from public cloud offerings. Insights shared in the article are from the research and practical experiences in self-managing the geographically distributed cluster and designing applications to use the Cassandra database efficiently.*

Keywords *- Availability, Cassandra, Distributed databases, NoSQL, Scalability.*

1. Introduction

Distributed NoSQL databases play a vital role in the modern digital application landscape to solve challenges associated with data growth, increasing user base across the globe, and providing reliable services to customers. Reliable IT systems are the backbone of any enterprise company to achieve its goals in delivering solutions to its customers in this competitive era. One of the most critical components in any IT application landscape is the database, and enterprises cannot afford it to go down for the normal application operation. Traditionally, single-server relational databases have fulfilled business needs, but scalability has always been challenging. With the growth of distributed databases, especially NoSQL databases, enterprises can now build solutions for the global user base without concern about availability and scalability challenges.

There are tens of distributed databases that deliver various benefits, and architects can choose one of these databases based on the application requirements. This article discusses the Apache Cassandra database, invented at Facebook [4] and made available for the open-source community.

There has been tremendous community growth around Apache Cassandra over the last decade, and a lot of top-tier companies [2] use this database to reap the benefits it offers. Cassandra can be self-hosted and managed by IT or used as a managed service from cloud offerings. Setting up a self-hosted Cassandra database requires infrastructure expertise, an understanding of advanced configurations and enterprise vendor support sometimes. However, using the Cassandra database in an enterprise still requires operations expertise, an understanding of key data modeling concepts and proper application design to use the database efficiently.

Much knowledge is available on the market for the initial infrastructure set up of the Cassandra database. However, the responsibility of designing a data model, designing applications, and operating it efficiently is still the responsibility of enterprises building applications for their customers. There is still a research gap in understanding important topics to achieve better performance with the Cassandra database, design data models, and choose proper client configurations to benefit from high availability. This article addresses the research gap by explaining important topics and how they affect the performance and availability of the Cassandra database and explains test results to show

that configurations explained in the article indeed help with improving performance and designing applications for high availability with the Cassandra database.

2. Literature Review

Apache Cassandra database has been around for more than a decade. Several research articles were written about the Apache Cassandra overview [1], applications of Cassandra [8][9][10], comparisons with other databases [5][6][7], and several books were written about advanced configurations and how to operate the cluster as well. However, there is still a lack of research from the application design and the enterprise usage perspective. Understanding the architecture and all the configurations and applying that knowledge in production takes a considerable amount of time, and enterprises often learn these important levers to tune after deploying the solution to production. This article provides insights on these important topics and how to address them before deploying the solution in production and achieving high availability and zero unscheduled downtimes.

3. Cassandra Architecture Overview

Cassandra is a distributed NoSQL database and a wide-column store [16]. It has a masterless architecture; no dedicated nodes act as masters, and all nodes in the cluster service requests. Cassandra can scale up to thousands of nodes horizontally using commodity hardware. Data from these nodes can be replicated geographically across multiple data centers. It replicates data to the node within the data center and across data centers as well. Cassandra's data model should be defined based on how the data is going to be queried from tables. Partition key is an important concept in Cassandra's data model, which will be explained in the later sections. The architecture discussion of distributed databases is incomplete without discussing the CAP theorem [3].

CAP refers to consistency, availability, and partition tolerance. None of the distributed databases provide all three guarantees. Cassandra prefers the availability and partition tolerance aspects of the CAP theorem. Cassandra prefers availability over consistency and provides partition tolerance. This is the reason Cassandra can be made highly available without incurring downtime. Cassandra writes data to the commit log (disk) first and then to Memtable, an in-memory data structure. Data from Memtables is flushed to the disk as SSTables once the Memtable threshold is reached or the draining process is executed explicitly. SSTables are the destination where the data lives in Cassandra. Commitlogs are useful in the event of a node crash or restart, as data from the commit log is replayed at the start of the node. Cassandra also provides easy backup and restore options in case of data recovery situations. A deeper understanding of Cassandra's architecture can help achieve scalability, availability, and resiliency with effort.

4. Important Considerations

This section describes important considerations in detail.

4.1. Partition Key

The partition key is an important concept in the entire Cassandra data modeling. Cassandra distributes data across the cluster using the hash of the partition key. One partition can have multiple rows. The clustering key allows us to contain multiple rows within a partition—combining the partition key and clustering key forms the primary key. The partition key should be chosen carefully to disperse data across the cluster. Otherwise, it can lead to hot partitions that can overload individual nodes in the cluster and affect the cluster's performance. One of the key things that should never be wrong when using Cassandra is the partition key.

4.2. Replication Factor

Cassandra replicates data across multiple nodes, and the replication factor defined on the keyspace dictates how many data replicas are maintained in the cluster. The higher the replication factor, the more data is available on more nodes, but it contributes to the disk growth quickly and demands adding more nodes quickly. The minimum replication factor should be 3 for any production application, and anything higher adds an advantage, but there is a cost associated with it. The minimum recommendation is 3 because there is less probability that 3 nodes in a cluster will be down simultaneously. Keyspace should be configured with NetworkTopologyStrategy for the replication to work [11].

4.3. Multiple Data Centers

Cassandra excels at replicating data seamlessly across multiple data centers. All data centers in Cassandra contain a set of nodes, and these data centers form one cluster. Datacenters can be in one geographic region or geographically distributed across different continents if network connectivity exists between these data centers. All data centers in the Cassandra cluster share the same schema, and data center replication settings are configured at the keyspace level. Having more than one data center is very important for the production cluster as it is easy to fail over if the current data center used by an application goes down for some unexpected reason.

4.4. Writes

Because of the masterless architecture of Cassandra, any node in the data center can take a request, which will become the coordinator node for a read or write. The coordinator node is responsible for sending writes to replica nodes in the cluster synchronously to some nodes and asynchronously to some other nodes, and this ratio of nodes is determined by the consistency level used while writing data from an application. Data replication also happens to remote datacenters and depends on the consistency level used for writes. Applications should be designed to reduce the latency of writes, which is possible if the replication to remote

datacenters happens asynchronously. If the replica node is not available at the time of writing, the coordinator node creates hint files, and these hints are kept for 3 hours on the coordinator node. This is why Cassandra is highly available for writes, as it continues to accept even though one of the primary replica nodes is down. If it takes over 3 hours to return the node, a repair operation should be done to replicate the missed data.

4.5. Reads

During reads, Cassandra makes the best effort to provide the latest available data from the replica nodes. As mentioned about the CAP theorem earlier, Cassandra prefers availability over consistency. The client may not receive the latest data if any of the replicas are down, which had the latest data. This is the reason Cassandra is highly available for reads as well. Cassandra read queries must always include the partition key. That way, the coordinator node calculates the hash and directs the query to the replica node where the data lives. There is an ALLOW FILTERING option to be used in the query without mentioning the partition key, which should never be used in production. With this option, the coordinator fetches data from all nodes for that table and applies the filtering to return the data back. If the table is big, these types of read queries affect the stability of the cluster. There is an option to create secondary indexes in Cassandra, but there will be a hit on the performance during writes as these indexes need to be maintained. These secondary indexes are local to each node, i.e., data available on the node is indexed locally, which means a read query using a secondary index should traverse all nodes to find the data, which can introduce latency. Suppose consumers need to read data after a few seconds, and there is no application requirement to read after write on non-partition key columns. In that case, the CDC option can be considered, which will replicate data from Cassandra to other databases such as Elasticsearch OpenSearch for search queries. This CDC option enables many other use cases for data consumers, and Cassandra continues to excel at providing write throughput and reads using partition keys for the optimal performance of the cluster.

4.6. Consistency Level

The replication factor on the keyspace and the consistency level chosen for writes and reads are key aspects in achieving high availability. Cassandra has several consistency levels [11] that can be tuned based on the application requirement. Assuming the replication factor is set to 3 and If LOCAL_QUORUM is used as the consistency level for writes, 2 nodes (*N/2 + 1*) should accept writes synchronously, and another node will get the write asynchronously. LOCAL_QUORUM writes data within the local data center, where the coordinator received the request. Another option is to use QUORUM, which calculates the number of nodes to write synchronously as (*total replication factor all DCs/2 + 1*). This can cause an issue with the availability of remote datacenters in other continents involved in the QUORUM. EACH_QUORUM can select a quorum in each data center if the datacenters are within a geographical region. This guarantees that data is available in all datacenters within a region. Writing with EACH_QUORUM is beneficial when data consumers are distributed in datacenters within a region. These consumers can read data with a consistency level as LOCAL_ONE from any of the datacenters, and data availability will be guaranteed. The same can be achieved when the LOCAL_QUORUM is used as the consistency level, but the read should be within the same data center with the LOCAL_ONE consistency level to guarantee data availability. Consistency level is the key lever to tune, which can shift the balance between consistency and availability of the CAP theorem. Other consistency levels exist, such as the number of nodes or ALL. This means the specified number of nodes or all nodes should accept writes, or if it is used for reads, many nodes should respond for reads. So, the greater the number of nodes involved in writes or reads, availability will be sacrificed, but higher consistency is guaranteed. This is where the consistency level used in the application and replication factor on the keyspace should be carefully chosen to balance between availability and consistency.

4.7. Repairs

Cassandra guarantees eventual consistency, which means data will be consistent eventually across the cluster. When there are no network issues and nodes are not down or not overloaded, data will replicate to replica nodes as it is supposed to. This is an ideal situation where everything works in perfect harmony, but that is practically impossible. There will be some network issues in the data center, and nodes may be down because of performance issues or scheduled maintenance. So, the Cassandra repair process triggers the comparison of the data across replicas [19] and streams the latest data from nodes to other nodes where there is stale data. Tools such as Cassandra Reaper schedule repairs automatically and use cluster resources efficiently. There are two types of repairs: incremental and full repairs. Incremental repairs should be done regularly, and full repairs can be done occasionally. Repairs are important to keep the data consistent across the cluster, so extra attention should be given to this.

4.8. Compaction

As mentioned in the architecture overview, data from Memtables gets flushed as SSTables to the disk. Cassandra SSTables are immutable, so once the data is written to the SSTable, its contents cannot be changed, but a new SSTable can be created by merging multiple SSTables. This process of merging SSTables is called compaction. A few compaction strategies are available in Cassandra, so depending on the data type and Time to Live configured on the table, the correct compaction strategy should be chosen for the optimal compaction. Compaction is a CPU-intensive

process; more threads can be assigned to complete compaction quickly if the CPU is available on the node.

4.9. Mutation Size

This size defines the max cell value in Cassandra, which is half the size of the commit log segment size, 32MB. By default, the mutation size is 16MB, so anything more than this size in a cell cannot be written to Cassandra. Attention should be given to the amount of data inserted in each cell. Suppose an application is required to insert a large data size in each cell. In that case, the clustering key can help to split the data into small chunks within a partition, but the application should stitch the data back once the data is retrieved for the entire partition. This way, the single partition can accommodate large data sizes. However, the general recommendation is to keep the partition size to 5 to 10MB for the optimal performance of the cluster [14].

4.10. Garbage Collection

Cassandra runs in JVM (Java virtual machine), so configurations should be tuned to allocate the right amount of Heap. Several JVM configurations can be tuned for writes and read [15]. It should be investigated if there are full garbage collection cycles or if garbage collection itself takes more than 100ms. For the efficient operation of the cluster, garbage collection should happen fast; otherwise, it is a sign that the cluster is overloaded and there are bad read queries pulling a lot of data. Garbage collection logs should be enabled in production to analyze them regularly to identify any bottlenecks, which gives an early sign if issues are building up.

4.11. Failover

Cassandra requires a contact point list on the client to establish a connection with the cluster. In Cassandra java driver version 3.x, DCAwareRoundRobinPolicy includes contact points from the local data center and configured number of servers per remote data center. So, if all nodes in the local data center are down, the client will try to connect to the remote data center. The problem with this approach is that if the remote DC is in a different geographical region and the consistency level is strict enough to require nodes from all datacenters to be available, the client will see issues performing read/write operations.

It has been recommended to have the failover at the infrastructure level, and Cassandra java driver version 4.x removed DCAwareRoundRobinPolicy and DefaultLoadBalancingPolicy is preferred to be used on the client side [13]. DefaultLoadBalancingPolicy lists nodes only from the local data center. The type of failover approach and consistency level dictates the application behavior if any data center were to go down. This topic should be given enough attention for production deployments, which dictates the application's availability in case of disasters.

4.12. Networking

Cassandra uses a gossip protocol to communicate the cluster state with other nodes in the cluster because of its distributed design. This is how nodes will be marked up or down in the cluster. Coordinator nodes communicate with other nodes in the cluster during read and write paths. Having a stable network connection between the nodes in the Cassandra cluster is very important for the healthy operation of the cluster. There can be network issues between clients and the Cassandra cluster itself. Sometimes, client-side errors can be misleading when some nodes are not reachable. In production, network issues between the client and the Cassandra cluster may raise false alarms, which can lead to the conclusion that the Cassandra cluster is unstable. So, it is important for node-to-node networking to function properly within the cluster and node-to-client connectivity is also functioning correctly.

4.13. Disk

Cassandra requires fast disk storage for the performance and reliability of the Cassandra database highly depend on the disk type used and configurations. By design, high write throughputs are supported by Cassandra, so the disk should support high I/O operations. At least 40 to 50% of the disk should be free on each node because the compaction process requires additional temp storage. If the nodes are reaching 60% of the disk capacity, it is time to add more nodes to the cluster to increase the disk capacity of the cluster. Proper monitoring must be in place to check the disk size, as the disk is one of the most important things in the Cassandra cluster and helps achieve the reliability and availability required for global applications.

4.14. Time to Live (TTL)

Time to live is an optional configuration in Cassandra that dictates how long data is going to be kept from the right time. Data will expire once the TTL is reached, and a tombstone will be created, but it will be available for clients to query until the GC grace period. TTL can be applied at the table level or during the write time. Which approach to use depends on the application requirement. However, it is preferable to set it at the table level as an application usually writes data to one table, making it easier to have consistent TTL for all records in a table. Expired data will be removed from SSTable during the compaction process.

4.15. Tombstones

Tombstone is a marker used in Cassandra to mark deletions or expired data. The compaction process removes tombstones from SSTables after the GC grace period. For the optimal operation of the cluster, having fewer tombstones is very important. Otherwise, performance issues will affect queries and availability of the application. Common pitfalls to avoid are writing null values in cells from the application and deleting queries.

Table 1. Test results using the cassandra stress test tool on a 3-node cassandra cluster

Read/Write	Replication Factor	Number of up nodes	Consistency Level	Ops/sec	Mean Latency (ms)	Total Time (secs)
Write	3	3	one	12,183	4.1 milli secs	82 secs
Write	3	3	local_quorum	11,511	4.3 milli secs	86 secs
Write	3	3	all	10,801	4.6 milli secs	92 secs
Write	3	2	one	14,842	3.3 milli secs	67 secs
Write	3	2	local_quorum	9,475	5.2 milli secs	105 secs
Write	3	2	all	Failure	Failure	Failure
Read	3	3	one	14,614	3.4 milli secs	68 secs
Read	3	3	local_quorum	9,820	5.1 milli secs	101 secs
Read	3	3	all	9,155	5.4 milli secs	109 secs
Read	3	2	one	13,404	3.7 milli secs	74 secs
Read	3	2	local_quorum	8,451	5.9 milli secs	118 secs
Read	3	2	all	Failure	Failure	Failure

4.16. Backups

Cassandra allows us to take a snapshot of a table at a point in time or take the incremental backup whenever Memtables are flushed to disk as SSTables. Snapshots and incremental backups can be paired to save disk space while taking backups. It is critical to take backups of the business-critical data or cluster backups. In case of data loss or corruption, it is beneficial to have a backup copy to restore it easily. Once backups are created using snapshot or incremental, backups can be moved to a location outside of the cluster to save disk space on the cluster.

4.17. Number of Nodes

The number of nodes in the cluster is an important factor in achieving the required performance. Peaking CPU and memory of the nodes continuously as an indicator to add more nodes to the cluster. Also, as mentioned earlier, if disks in the cluster reach 60% of the capacity, new nodes should be added to increase the cluster capacity. If clients report slowness or failures with reads and write intermittently and the network seems stable in the data center, it is time to scale the cluster. Proper planning should be done in advance to add more nodes if there is an expected demand soon, otherwise putting more load on the cluster will affect the overall performance of the Cassandra database and affect availability.

4.18. Monitoring and Metrics

Finally, Cassandra provides various metrics [18], which can be scraped with Prometheus and visualized in Grafana, and monitors can be setup on these metrics. It is important to measure and observe write loads across the cluster, GC durations, write and rate latencies, Memtable sizes and Cassandra table sizes with these metrics and decide on scaling the cluster or tuning the cluster for optimal performance.

5. Results and Discussion

Tests were conducted on a Cassandra 3-node cluster; each node has 4 CPUs, 16GB memory and 500GB disk. Cassandra stress tool [17] was used to test writes and reads with 50 threads; 1 million records are used in each test, with varied levels of consistency, keeping all nodes up and bringing one down in the cluster as well. Keyspace was created in advance, and the replication factor was set to 3 to replicate data across 3 nodes in the cluster. It is clear from the test results listed in Table 1 that the write and read performance decreases with stronger consistency. Table 1 contains various combinations' results that provide insights into the client and Cassandra's behavior. As shown in Figure 1, write operations per second are reduced as the consistency level increases from one to all (weaker to stronger consistency). This is the reason consistency should be tuned properly for better performance. When one node is down in the cluster, write performance still follows the downward trend with the stronger consistency level, but with all consistency levels, write operations count came down to 0. This explains that all consistency levels, which are the strongest, make the database unavailable for application even with one node down in the cluster. This also justifies the CAP theorem, as consistency and availability act in the opposite direction.

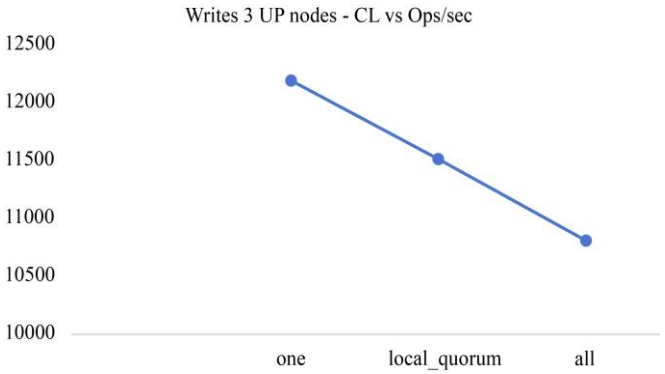

Fig. 1 Writes with all nodes up in the cluster

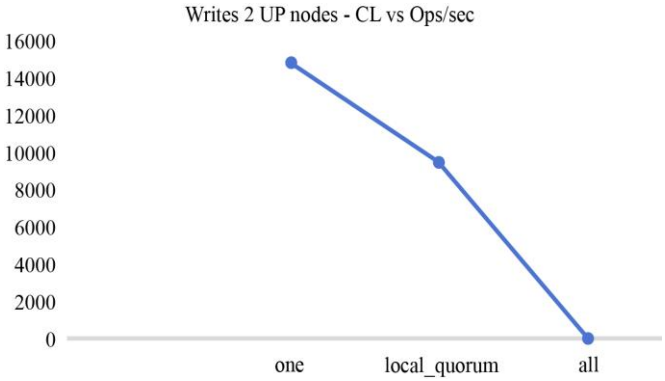

Fig. 2 Writes with one node down in the cluster

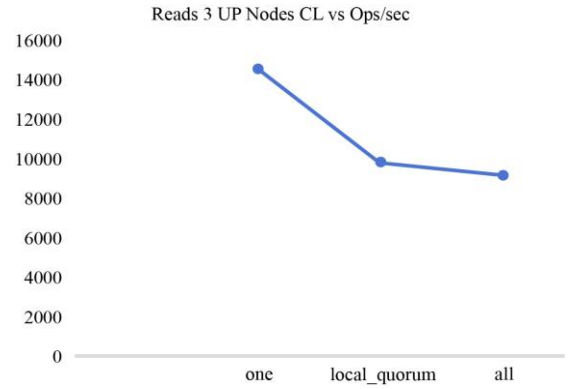

Fig. 3 Reads with all nodes up the cluster

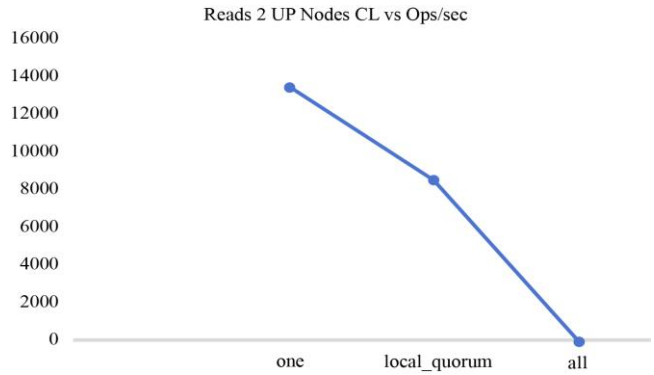

Fig. 4 Reads with one node down in the cluster

Similar performance results are observed with reading tests as well. As the consistency level is increased, there is a decrease in operations per second. With the strongest consistency, when one node is down, the read operation count is 0, resembling an application outage. These tests conclude the key aspect of the Cassandra database, consistency vs. availability and how it affects the availability of an application. Also, it shows how Cassandra continues to serve application requests even when one of the nodes is down in the cluster.

6. Conclusion

The above-listed important topics and observations from the test results will help to take steps to achieve optimal performance. However, the Cassandra database has many other configurations that can be tuned based on operational needs. Proper monitoring should be in place to observe key performance metrics, and continuous tuning is required to achieve better performance. As with any other production system, the more we invest in improving reliability availability, the better the results will be.

Funding Statement

There was no external funding obtained to prepare this article. This article was written in the author's personal time and practical experience gained while working on the Cassandra database. Tests were conducted on the Cassandra cluster setup in the GCP cloud with the author's funding.

References

[1] Abdul Wahid, and Kanupriya Kashyap, "Cassandra-A Distributed Database System: An Overview," *Emerging Technologies in Data Mining and Information Security, Advances in Intelligent Systems and Computing*, vol. 755, pp. 519-526, 2019. [CrossRef] [Google Scholar] [Publisher Link]

[2] Pedro Martins et al., *NoSQL Comparative Performance Study*, Trends and Applications in Information Systems and Technologies, WorldCIST 2021, Advances in Intelligent Systems and Computing, vol. 1366, pp. 428-438, 2021. [CrossRef] [Google Scholar] [Publisher Link]

[3] Guoxi Wang, and Jianfeng Tang, "The NoSQL Principles and Basic Application of Cassandra Model," *2012 International Conference on Computer Science and Service System*, pp. 1332-1335, 2012. [CrossRef] [Google Scholar] [Publisher Link]

[4] Avinash Lakshman, and Prashant Malik, "Cassandra: A Decentralized Structured Storage System," *ACM SIGOPS Operating Systems Review*, vol. 44, no. 2, pp. 35-40, 2010. [CrossRef] [Google Scholar] [Publisher Link]

[5] Muh. Rafif Murazza, and Arif Nurwidyantoro, "Cassandra and SQL Database Comparison for Near Real-Time Twitter Data Warehouse," *2016 International Seminar on Intelligent Technology and Its Applications (ISITIA)*, pp. 195-200, 2016. [CrossRef] [Google Scholar] [Publisher Link]

[6] Giuseppe Baruffa et al., "Comparison of MongoDB and Cassandra Databases for Spectrum Monitoring As-a-Service," *IEEE Transactions on Network and Service Management*, vol. 17, no. 1, pp. 346-360, 2020. [CrossRef] [Google Scholar] [Publisher Link]

[7] Vishal Dilipbhai Jogi, and Ashay Sinha, "Performance Evaluation of MySQL, Cassandra and HBase for Heavy Write Operation," *2016 3rd International Conference on Recent Advances in Information Technology (RAIT)*, pp. 586-590, 2016. [CrossRef] [Google Scholar] [Publisher Link]

[8] Shubham Dhingra et al., "Fault Tolerant Streaming of Live News Using Multi-Node Cassandra," *2017 Tenth International Conference on Contemporary Computing (IC3)*, pp. 1-5, 2017. [CrossRef] [Google Scholar] [Publisher Link]

[9] Katalin Ferencz, and József Domokos, "IoT Sensor Data Acquisition and Storage System Using Raspberry Pi and Apache Cassandra," *2018 International IEEE Conference and Workshop in Óbuda on Electrical and Power Engineering (CANDO-EPE)*, pp. 143-146, 2018. [CrossRef] [Google Scholar] [Publisher Link]

[10] Gautam Pal, Gangmin Li, and Katie Atkinson, "Near Real-Time Big Data Stream Processing Platform Using Cassandra," *2018 4th International Conference for Convergence in Technology (I2CT)*, pp. 1-7, 2018. [CrossRef] [Google Scholar] [Publisher Link]

[11] Cassandra Documentation, Ensure Keyspaces are Created with Network Topology Strategy, Apache Cassandra. [Online]. Available: https://cassandra.apache.org/doc/4.1/cassandra/getting_started/production.html#ensure-keyspaces-are-created-with-networktopologystrategy

[12] Cassandra Documentation, Tunable Consistency, Apache Cassandra. [Online]. Available: https://cassandra.apache.org/doc/latest/cassandra/architecture/dynamo.html#tunable-consistency

[13] Load Balancing, DataStax Documentation. [Online]. Available: https://docs.datastax.com/en/developer/java-driver/4.2/manual/core/load_balancing/

[14] Cassandra Optimal Partition Size, Stackoverflow. [Online]. Available: https://stackoverflow.com/questions/69282435/cassandra-optimal-partition-size

[15] Garbage Collection Tuning for Apache Cassandra, The Last Pickle, 2018. [Online]. Available: https://thelastpickle.com/blog/2018/04/11/gc-tuning.html

[16] Wide-Column Store, Wikipedia. [Online]. Available: https://en.wikipedia.org/wiki/Wide-column_store

[17] Cassandra Stress, Cassandra Documentation, Apache Cassandra. [Online]. Available: https://cassandra.apache.org/doc/latest/cassandra/tools/cassandra_stress.html

[18] Cassandra, Operating, Metrics, Cassandra Documentation, Apache Cassandra. [Online]. Available: https://cassandra.apache.org/doc/latest/cassandra/operating/metrics.html

[19] Repair, Cassandra Documentation, Apache Cassandra. [Online]. Available: https://cassandra.apache.org/doc/latest/cassandra/operating/repair.html