THE JOURNAL OF DIGITAL FORENSICS, SECURITY AND LAW

Journal of Digital Forensics, Security and Law

Volume 15 Article 5

February 2021

Backup and Recovery Mechanisms of Cassandra Database: A Review

Karina Bohora
College of Engineering, Pune, bohoraka16.it@coep.ac.in

Amol Bothe
College of Engineering, Pune, botheap16.it@coep.ac.in

Damini Sheth
College of Engineering, Pune, daminihs16.it@coep.ac.in

Rupali Chopade
College of Engineering, Pune, rmc18.comp@coep.ac.in

V. K. Pachghare
College of Engineering, Pune, vkp.comp@coep.ac.in

Recommended Citation

Bohora, Karina; Bothe, Amol; Sheth, Damini; Chopade, Rupali; and Pachghare, V. K. (2021) "Backup and Recovery Mechanisms of Cassandra Database: A Review," *Journal of Digital Forensics, Security and Law*: Vol. 15 , Article 5.
DOI: https://doi.org/10.15394/jdfsl.2021.1613
Available at: https://commons.erau.edu/jdfsl/vol15/iss2/5

BACKUP AND RECOVERY MECHANISMS OF CASSANDRA DATABASE: A REVIEW

Karina Bohora, Amol Bothe, Damini Sheth, Rupali Chopade, V. K. Pachghare
College of Engineering, Pune
bohoraka16.it,botheap16.it,daminihs16.it,rmc18.comp,vkp.comp@coep.ac.in

ABSTRACT

Cassandra is a NoSQL database having a peer-to-peer, ring-type architecture. Cassandra offers fault-tolerance, data replication for higher availability as well as ensures no single point of failure. Given that Cassandra is a NoSQL database, it is evident that it lacks research that has gone into comparatively older and more widely and broadly used SQL databases. Cassandra's growing popularity in recent times gives rise to the need to address any security-related or recovery-related concerns associated with its usage. This review paper discusses Cassandra's existing deletion mechanism and presents some identified issues related to backup and recovery in the Cassandra database. Further, failure detection and handling of failures such as node failure or data center failure have been explored in the paper. In addition, several possible solutions to address backup and recovery, including recovery in case of disasters, have been reviewed.

Keywords: Cassandra, NoSQL, Data Recovery, Backup, Node Failure

1. INTRODUCTION

1.1 Architecture and Working

Cassandra's data structure's basic component is the column, which consists of a (key, value) pair for each row that combines to form a partition key. As shown in Figure 1, partitioning of data is done based on its partition key which is passed through a hash function. Data with the same partition key will be stored on the same node within the cluster.

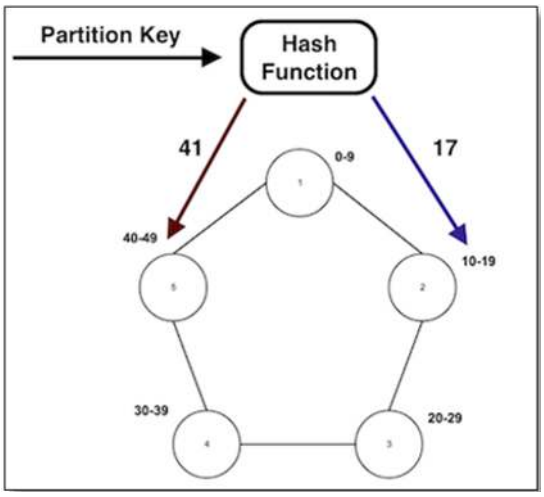

Figure 1. Partition Key Mechanism

Figure 2 shows that when a read operation is called, it first searches in the MemTable (*Problems we see in support is data going "missing". | Datastax*, 2020), which is the cache for the data in the RAM. If it is unable to find, then it checks in the SStable, which is

on the disk. When a write operation is called,

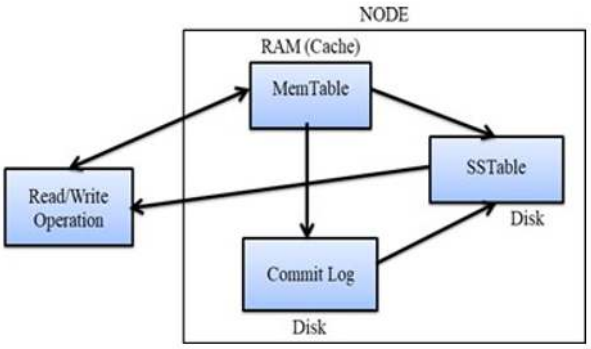

Figure 2. Architecture of a Node

it writes in the MemTable and to the commit log for durability. Once the MemTable is full, it flushes the data to the SSTable and is permanently written there.

There is more than 1 replication factor in Cassandra, and hence, there's high availability and no single point of failure. It has the property of 'Linear scale performance,' wherein as more nodes are added, the performance of Cassandra increases. The data is automatically replicated to multiple nodes for fault-tolerance. Replication across multiple data centers is supported.

1.2 Existing Provision for Security in Cassandra

For immediate deletion, i.e., without waiting for any period of time, Cassandra supports the *'DROP KEYSPACE'* and *'DROP TABLE' statements* (*Problems we see in support is data going "missing". | Datastax*, 2020). Besides these two statements, there are two other methods of deletion, as shown in Figure 3, in which immediate deletion does not take place. These methods are:

1. user issues a delete command

2. user marks record (row/column) with TTL (time to live).

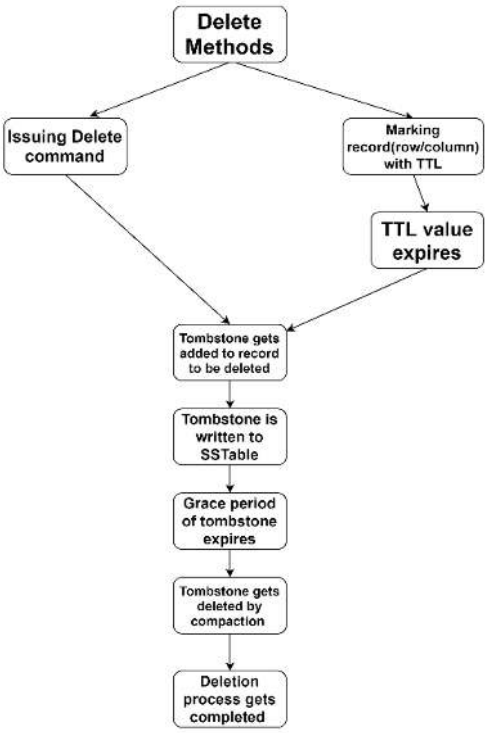

Figure 3. Delete Methods

In the 1st method, when the delete command is issued, a tombstone, which marks the record for deletion, gets added to a particular record. The tombstone is then written to the SSTable. When the built-in grace period (expressed as gc_grace_seconds) expires, the tombstone is deleted. The default value for grace period is 864,000 seconds (ten days) (*Apache Cassandra 3.0 for DSE 5.0. How is Data Deleted*, 2020). However, each table can have its own value for the grace period. In order to identify the grace period of a tombstone, the *'cassandra.yaml'* file can be used. The *'cassandra.yaml'* file consists of configuration properties, one of which is the *'gc_grace_seconds.'* The value in *'gc_grace_seconds'* denotes the grace period associated with a tombstone.

In the 2nd method, the user marks row/column with a TTL value. In order to identify the TTL associated with a record, one can use the 'TTL function'. The TTL function takes

1 argument, which is a column name, and returns the corresponding associated TTL. After the TTL value expires, that particular record is marked with a tombstone, and then this tombstone is written to SSTable. Then, the tombstone is deleted when the grace period expires.

In order to identify whether a record has a tombstone associated with it, it is required to check if the associated SSTable dump output of the corresponding partition has the *'deletion_ info'* tombstone marker. The *'deletion_ info'* tombstone marker present in the SSTable dump output includes a *'marked_ deleted'* timestamp which indicates the time at which a record was requested to be deleted as well as a *'local_ delete_ time'* timestamp indicating the local time at which the Cassandra server received a request to delete a record. Hence, the *'marked_ deleted'* timestamp is specified by the user or the user application, and the 'local_delete_time' timestamp is set by the Cassandra server.

The SSTable event log dump has the *'deletion_ info'* associated with a deleted record. To read the log dump of SSTable, *'cassandra-tools'* can be installed, and *'nodetool'* can then be used to create a snapshot, and consequently, an *'sstabledump'* can be generated and stored in a file. An analyst can access this log dump to identify information pertaining to the deletion of a record.

There are 3 main reasons why the backup of data is very necessary:

(a) due to defect in application logic, good data might get overwritten and replicated across nodes

(b) SSTables can become corrupted

(c) a disaster recovery plan will be of no use if the multi-data center failure occurs.

Cassandra has two mechanisms of backup and three types of backup (*Problems we see in support is data going "missing". | Datastax*, 2020). The backup mechanisms are

(a) snapshot

(b) incremental backup.

The main difference between these two is that snapshots provide full backup, while incremental backup includes changes made for a period of time. The types of backup are: full, incremental, and differential.

The difference between incremental backup and differential backup is that incremental backup includes all changes since the previous incremental backup, while differential backup has all changes since the previous full backup. Basically, a series of incremental backups together constitute a differential backup. As mentioned, Cassandra has a provision for backup and restore but lacks data recovery of missing/deleted data (*Apache Cassandra 3.0 for DSE 5.0. Backing up and Restoring Data| Datastax*, 2020). Since it operates with hundreds of thousands of nodes spread across data centers, it will be common for several small and large components to fail simultaneously, and so they must be treated as a norm and not an exception. If Cassandra can recover data successfully, then it will improve the system's reliability, performance, efficiency, and scalability.

2. FAILURE HANDLING, BACKUP & RECOVERY

As thoroughly explored in (Wang & Tang, 2012), NoSQL databases such as Cassandra are based on the theories of CAP theorem, BASE theorem as well as the Eventual consistency theorem. One of the biggest challenges that NoSQL deployments, including Cassandra, face is data protection and retention. Even the replicas get damaged in the cases of

ransomware attacks, data corruption, or accidental deletion. For NoSQL databases, the key challenges in backup and restore include cluster-wide consistent backup, removing redundant copies, and being resilient to topology changes. Cassandra effectively addresses agility and scaling required in new-age applications such as IoT, which has caused times series data to become more common. The paper (Arous et al., 2019) shows how this time series data can be recovered using RecovDB, a system that is enhanced to support the recovery of multiple blocks. It can be used for Online Trading System applications, as demonstrated in (Wang & Tang, 2012). Cassandra can also be used to manage Big Data, as explained in (Mangle & Sambhare, 2013).

2.1 Failure Detection of One Node by Another Node in the System

Detecting the failure of a node in the system or detecting whether a node is up or down is crucial because if failure detection is not implemented, a node might try to communicate with unreachable nodes while operating. In Cassandra, for detecting failures of nodes, the use of a modified version of the ϕ Accrual Failure Detector is suggested. This module (Lakshman & Malik, 2010) is based on the idea that every node that is monitored will have a suspicion level associated with it instead of a plain Boolean value (0 if node is down and 1 if node is up). This suspicion level value associated with each of the monitored nodes is called ϕ. The value that ϕ takes on depends on the amount of load at that particular node and other network conditions. In simpler words, ϕ is a threshold value that tells us about the likelihood of making a mistake in marking a node as either up or down. The more the value of ϕ, the less likely it is that we are making a mistake in our decision. A value of ϕ=1 indicates that there is a chance of 10% that our decision will be contradicted in the future. Likewise, ϕ=2 and ϕ=3 indicate a likelihood of 1% and 0.1%, respectively for error in the decision. An important aspect in this whole method of Accrual Failure Detector is determining the value of ϕ. To determine the value of , the inter-arrival times of gossip messages from other nodes to a given node are analyzed. Figure 4, shows the existing as well as the ϕ Accrual Failure Detector method. It is observed and inferred that the distribution of these inter-arrival times can be better approximated as an Exponential Distribution as compared to a Gaussian distribution.

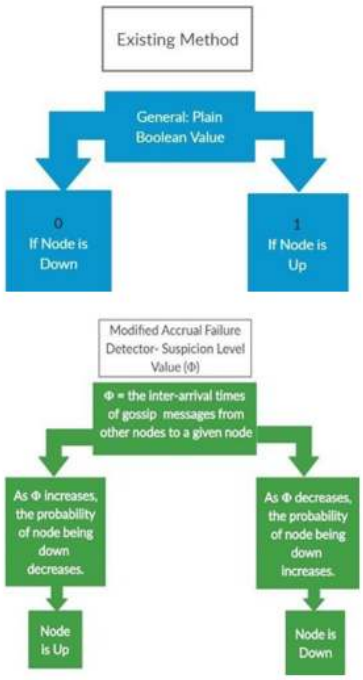

Figure 4. Failure Detection of One Node by Another

2.2 Handling Node Failure in the System

When any node starts for the first time in the ring architecture, it chooses its position by selecting a random token. This information

of mapping regarding which node is mapped to which token is stored on the local disk and is gossiped in the cluster so that all the nodes come to know about the mapping. So each node knows about the positions of all the other nodes in the ring topology. Each data item stored has a key associated with it. N hosts or nodes have a replica of a given data item. N is called the replication factor. Each key has a coordinator node assigned to it. The coordinator node stores a copy of the key with itself and further replicates it to N-1 nodes. So, in all, N nodes have the data item with them. In Cassandra, there are three main replication policies for replicating data items across nodes. These are:

1. Rack Aware
2. Rack Unaware
3. Datacenter Aware

In the Rack Unaware strategy, the N-1 successors of the coordinator node are chosen as non-coordinator replicas for the data item. On the other hand, in both Rack Aware and Datacenter Aware strategies, a leader called Zookeeper is elected. This leader or Zookeeper tells the nodes the ranges up to which they will be acting as a replica for. So each node has ranges for which it is responsible. The Zookeeper tells each node about the ranges for which it is responsible. The Zookeeper takes care that a node is responsible for a maximum of N-1 ranges. This means that no node in the system can have more than *N-1* ranges associated with it. The information regarding all the ranges for which all the nodes in the system are responsible for, is stored inside the Zookeeper. So, when a node fails or crashes and loses the information about the ranges for which it was responsible, then the Zookeeper node comes to its rescue and provides it the required ranges for which it is responsible when the node comes back up or recovers from the crash. In this way, the Zookeeper node can help a crashed node get back lost information about the ranges for which it is responsible for holding replicas.

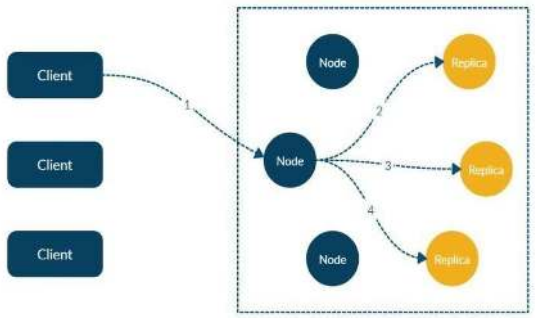

Figure 5. Data Replication to Handle Data Center Failure

2.3 Handling Data Center Failure in System

The main reasons for the occurrence of data center failure are natural disasters, network failure, and power outages. In Cassandra (Qiao et al., 2018) each row is replicated across many data centers. Each row has a partition key associated with it, and each key has a preference list that mentions the nodes across which that particular row should be replicated. These nodes are spread across different data centers. This ensures that if an entire data center fails, there will still be a replica of the row with some other node/s in different data center/s. Figure 5 shows the replication of data across nodes belonging to different data centers. Different data centers are connected via high-speed network links.

2.4 Trade-Off between Data Recovery Time and Query Latency

While designing the database schema to be followed, it is crucial to know the different potential query patterns to ensure that query latency is low and that throughput and data

recovery are high. For this, a scheme of heterogeneous replica instead of a homogenous replica is suggested in (Qiao et al., 2018). Various replicas will have the same datasets in this scheme but can have different serialization bytes on disk. As a result, different types of queries will get accelerated while the data recovery ability of original data is maintained. It attempts to improve Cassandra's query performance by introducing a heterogeneous replica mechanism for replicating data while maintaining the original data recovery properties for which the idea of replication was initially introduced. For this, an algorithm called HRCA is proposed, which is able to improve the average query latency by 1 to 2 orders of magnitude. In this proposed method, different queries use different replicas, and so besides being useful for data recovery, the replicas can also be used to reduce query latency and improve query performance. If 'n' is the replication factor and there are 'm; clustering keys then there are *Cn m!+n1* possible layouts for the replica. This number is very large, and so only optimal heterogeneous replicas are considered based on simulated annealing. Here, since replicas are different, the LSM-Tree write process is used for data recovery as the original method applies only in the case of homogenous replicas while here heterogeneous replicas have been introduced. As query latency's issue was resolved, however, it was observed that data recovery took longer. The data that traditionally took 4 minutes to recover now took 6 minutes under this newly proposed method using heterogeneous replicas. This method, thus, demonstrates a clear trade-off that exists between query latency and data recovery time.

2.5 BARNS Solution for Cluster Consistent, Storage Efficient Topology Oblivious Backup

The BARNS solution proposed in (Kathpal & Sehgal, 2017) saves 66% backup space under a replication factor of 3 and has a constant restore time of 2 to 3 minutes. The time it needs for restoring is independent of cluster size and the dataset for which it is creating a backup. The BARNS solution uses lightweight snapshots instead of copy-based backup. This solution ensures resilience to topology changes during backup and recovery. This solution is proposed for Cassandra when it is hosted on shared storage. The most important and differentiating factor in the BARNS solution is that it performs recovery-related work at the backup time instead of during restoration. This method uses Cassandra's compaction feature to cluster-wide consistency and space efficiency. For achieving topology oblivious backup, this method saves cluster configuration for each cluster. This cluster configuration includes the health and token of each node in the specific cluster. In BARNS, the backup takes place in 2 phases:

1. Lightweight backup (LWB): Here, cluster configuration is saved, and a snapshot of healthy nodes is taken.

2. Post-processing phase (PP): In this phase, consistency conflicts are resolved, and redundant copies of data are removed by post-processing snapshots taken in 1st phase.

Figure 6 shows the pictorial representation of the two phases of backup as proposed by BARNS solution. It was observed that recovery took 60-80 seconds, and this time was independent of data as well as cluster size.

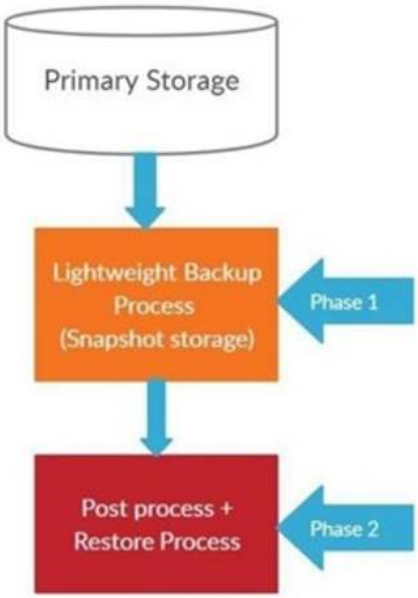

Figure 6. BARNS: 2 Phases of Backup

The *'nodetool repair'* command takes 456 seconds for the repair operation. So, BARNS demonstrates considerable improvement.

2.6 Decentralized Disaster Recover Centre (DR) Instead of Centralized DR

In a decentralized disaster recovery structure (Cankaya & Kupka, 2016), many servers host the database and each server caters to a different schema. In the proposed idea, each schema will have a different replication scheme. Furthermore, instead of a radial backup, a cyclic backup scheme is proposed for better decentralization. A DC is a data center, while a DR is a data recovery center. The biggest problem with having a single disaster recovery center is that it poses the threat of a single point of failure. In case of disaster, the DR will act as DC, and so it has to be adequately resourced. However, this increases the number of unused resources as the DR will remain idle for a lot of time. Two main proposals of this method are:

- Decentralized Disaster Recovery Centers: A database has different schema. In this method, groups of schemas are made, and each group is hosted on a different server.

- Conditional Forwarding of Queries to Idle DR Server: Here, the idea is to utilize idle CPU time and unused resources of DR servers. When DC is overloaded with a huge number of queries, some queries are forwarded to DR, and it handles them. Because DR dealing with updates as well as insertion queries may cause data inconsistency, a Controller module is introduced, which takes care of this issue.

2.7 Disaster Recovery by Replication to a Stable Storage

The incremental backup approach is supported by a few NoSQL databases but it lacks disaster recovery. To achieve this, a comparison of the local replicates with remote replicates is done, which is an expensive operation and costs overhead as it requires the transfer of the entire dataset. The general disaster recovery approaches are i) data replication and ii) periodic snapshot-based backup which periodically replicates the entire database to external data storage. These solutions are beneficial to minimize data loss, but it does not recover the latest copy data. In the proposed solution (Abadi et al., 2016), of trigger-based backup, whenever a new change is made in a document, it is immediately replicated to a stable storage. The document is saved with the document version number indicating which version of the original document it is with the latest modifications. This approach includes four modules:

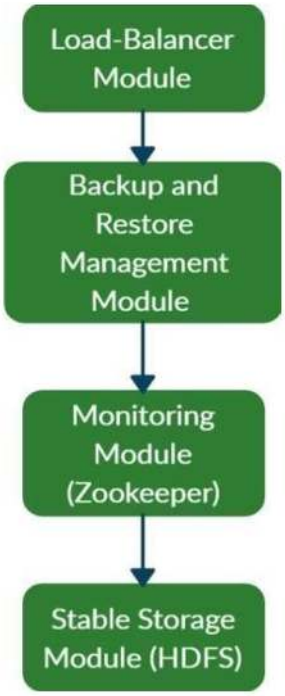

Figure 7. Disaster Recovery by Replication to a Stable Storage

(a) the load-balancer module that handles all the requests and forwards them to the next module;

(b) the backup and restore management module, which executes the operation described in the request received from the load-balancer tier;

(c) the monitoring module, implemented using ZooKeeper which selects the primary load- balancer in case of failure and

(d) the stable storage module, implemented using Hadoop File System (HDFS), in to which the data is replicated.

These four modules used to recover data by replicating it to a stable storage are shown in Figure 7.

2.8 Understanding of NoSQL Database

In today's scenario (Prasad & Gohil, 2014), web applications are facing new challenges while serving millions of users. It can be easily understood and appreciated that while accessing these web applications worldwide, users expect the services to be always available with high performance and reliability. The rate of growth of successful web services is much faster as compared to the increase in the performance of computer hardware. The NoSQL databases have a dynamic schema and are best suited for hierarchical data storage. NoSQL Database uses schema of a weaker BASE (Basic Availability, Soft state, eventual consistency) features as compared to a relational database which uses ACID (Atomic, Consistent, Isolated, Durable) properties. Table 1 shows the comparison of five NoSQL databases based on the properties of Replication, Consistency, and Partial tolerance. A distributed computer system can provide more computing power, and hence parallel computation (Bhattacharya et al., 2017). As suggested in (Mangle & Sambhare, 2013), Big Data is characterized by the amount of data generated (volume), the rate at which data is generated (speed), and the heterogeneity of data sources (variety). Hence, Big Data poses challenges to data management. Traditional RDBMSs are used for the management of structured data. By shifting towards NoSQL databases from SQL databases, an advantage of being able to work with semi-structured or unstructured data is achieved. Also, (Mangle & Sambhare, 2013) suggest different advantages of NoSQL databases over SQL databases. MongoDB and Riak are taken as two representative NoSQL databases, and their performance is evaluated with respect to the read and update operations.

Table 1. Comparison of NoSQL Databases

	Replication	**Consistency**	**Partial Tolerance**
SimpleDB	Yes	No	No
Cassandra	Yes	No	Yes
MongoDB	Yes	Yes	Yes
Redis	Yes (Unidirectional)	Yes	Yes
BigTable	Yes	Yes	Yes

2.9 Database Extraction Tools

There are quite a few tools that help in forensic analysis of databases ranging from backup and replication to data extraction. A survey of ten forensic data tools in (Cankaya & Kupka, 2016) provides insights on how each one of them is useful, listing out their main functionalities. The tools are Oxygen Forensics Detective, Xplico, Digital Detective Blade v1.13, Kernel Database Recovery, Systools SQL Log Analyzer, WinHex, NetCat, Windows Forensic Toolchest, SQL CMD, and Forensics Toolkit (FTK). These tools (Chopade & Pachghare, 2019) are either focused on data extraction or database recovery. A standard database is used to compare the performance by testing the runtime of similar tools. According to the tests in (Cankaya & Kupka, 2016), Forensics Toolkit (FTK) has the fastest runtime for static data compared to the other two tested against for forensic analysis.

2.10 Accurate and Efficient Missing Blocks Recovery for Large Time Series

The research in (Arous et al., 2019), deals primarily with missing value recovery. The major reasons for missing values are cited to be failures as well as irregular time intervals. The paper is focused on RecovDB, a relational database system. RecovDB has several advantages, including parameter-free recovery, correlation-aware recovery, and full-fledged DBMS support. There are three important properties of RecovDB, which are:

(a) recovering multiple time series at once,

(b) accurate recovery with increasing missing values

(c) efficient recovery with increasing data size. To measure the efficiency and accuracy of RecovDB,

2 criteria were used:

(a) runtime to perform full recovery

(b) accuracy of recovery using RMSE=root mean square error.

3. CONCLUSION

Recovery using multiple servers would fetch better results compared to traditionally used backup based on only a single server. To make the solution more realistic, we must consider network traffic and load balancing issues as well. If we use a trigger-based backup mechanism that gets triggered upon each database change, we will get lower RPO

(Recovery Point Objective) and RTO (Recovery Time Objective) values compared to the values we would have got by using periodic backup approaches. There is a trade-off between the query latency and data recovery time and an attempt to decrease query latency. It will most likely cause data recovery to take longer than usual, as demonstrated in the method of heterogeneous replicas for decreasing query latency. However, there does not necessarily have to be a trade-off between accuracy and efficiency while recovering time series data. There is an evident research gap for recovery in Database Cassandra as most of the tools help recover from SQL scripts that come under relational database, unlike Cassandra. The proposed solution of the 4-step trigger-based backup is promising as whenever a new change is made; it is immediately replicated to stable storage. The only downside to this technique is the excessive amount of storage space required due to multiple versions of the same document being replicated to a module. There is scope for future work on the BARNS solution by introducing incremental backup as it currently provides only full backup. The BARNS solution can also further be improved by reducing the post-processing times for Cassandra. Also, a trigger-based backup approach can be adopted for Disaster Recovery for Cassandra.

REFERENCES

Abadi, A., Haib, A., Melamed, R., Nassar, A., Shribman, A., & Yasin, H. (2016). Holistic disaster recovery approach for big data nosql workloads. In *2016 ieee international conference on big data (big data)* (pp. 2075–2080).

Apache cassandra 3.0 for dse 5.0. backing up and restoring data| datastax. (2020). `https://docs.datastax.com/en/archived/cassandra/3.0/cassandra/operations/opsBackupRestore.html`.

Apache cassandra 3.0 for dse 5.0. how is data deleted. (2020). `https://docs.datastax.com/en/dse/5.1/dse-arch/datastax_enterprise/dbInternals/dbIntAboutDeletes.html`.

Arous, I., Khayati, M., Cudré-Mauroux, P., Zhang, Y., Kersten, M., & Stalinlov, S. (2019). Recovdb: Accurate and efficient missing blocks recovery for large time series. In *2019 ieee 35th international conference on data engineering (icde)* (pp. 1976–1979).

Bhattacharya, S., Roy, A., Sen, S., & Debnath, N. C. (2017). Distributed data recovery architecture based on schema segregation. In *2017 ieee international conference on industrial technology (icit)* (pp. 1238–1243).

Cankaya, E. C., & Kupka, B. (2016). A survey of digital forensics tools for database extraction. In *2016 future technologies conference (ftc)* (pp. 1014–1019).

Chopade, R., & Pachghare, V. K. (2019). Ten years of critical review on database forensics research. *Digital Investigation*, *29*, 180–197.

Kathpal, A., & Sehgal, P. (2017). {BARNS}: Towards building backup and recovery for nosql databases. In *9th {USENIX} workshop on hot topics in storage and file systems (hotstorage 17)*.

Lakshman, A., & Malik, P. (2010). Cassandra: a decentralized structured storage system. *ACM SIGOPS Operating Systems Review*, *44*(2), 35–40.

Mangle, N., & Sambhare, P. B. (2013). A review on big data management and nosql databases in digital forensics. *International Journal of Science and*

Research (IJSR) Volume, *4*.

Prasad, A., & Gohil, B. N. (2014). A comparative study of nosql databases. *International Journal Of Advanced Research In Computer Science*, *5*(5).

Problems we see in support is data going "missing". | datastax. (2020). `https://academy.datastax.com/support-blog/dude-where\%E2\%80\%99s-my-data.`

Qiao, J., Huang, X., Rui, L., & Wang, J. (2018). Heterogeneous replica for query on cassandra. *arXiv preprint arXiv:1810.01037*.

Wang, G., & Tang, J. (2012). The nosql principles and basic application of cassandra model. In *2012 international conference on computer science and service system* (pp. 1332–1335).

