EFFECTIVE HACommit ARCHITECTURE FOR CLOUD DATA TRANSACTIONS BASED ON MEMORY CAPACITY AND ACTIVE CONNECTIONS TO HANDLE READ-WRITE TRANSACTIONS

A.Aasha Begum
Part-Time Research Scholar
Dept of Computer Science
Madurai Kamaraj University
Madurai

Dr.K.Chitra
Assistant Professor
Dept of Computer Science
Govt Arts College, Melur
Madurai – 625 106

Abstract

Cloud Computing enables convenient, on-demand network data access to shared configurable computing resources such as networks, servers, storage, applications, and services that can be rapidly provisioned and released with minimal management effort or service provider interaction. There are four basic models of cloud service: Infrastructure as a service (IaaS), Platform as a service (PaaS), Software as a service (SaaS), and Database as a Service (DaaS). The public Internet spawned private corporate intranets; cloud computing is now generating individual cloud platforms. Both public and different cloud platforms are looking to deliver the benefits of cloud computing to their customers. Whether the private or public cloud provides the Database is a critical part of that Platform. Therefore your cloud database must be compatible with cloud computing. One of the core design principles is dynamic scalability, or the ability to provision and decommission among the clients and servers on demand. Unfortunately, the majority of today's database servers are incapable of satisfying this requirement. Business applications also demand that the cloud database be ACID compliant: providing Atomicity, Consistency, Isolation, and Durability. In this paper, proposes the HACommit architecture (Highly Available Commit) evolves the cloud DBMS environment through the data transactions with the compelling read and write transactions of data: a highly-available data store that depends on the (Consensus Algorithm) consensus value of all active connections of the commit nodes. The client, whether commit or abort the transaction like decision-making approach while before to vote for a commit along with the transaction participant commit nodes through the clients and servers. The replication of data enables the consistent flow of the client's read-write logging and the server's read-write logging with the distributed commit process through the error-fault tolerance of timeout of transactions. The data transaction is easy to access to the other distributed commit data transaction. The client initiates the transaction commit, within one communication like as one phase of data transactions.

Key Words: Cloud computing, Cloud DBMS, HACommit (Highly Available Commit) architecture, Consensus Algorithm.

I. INTRODUCTION

In recent years, the cloud infrastructure highly available in web-based applications. The database transaction provides the optimal data storage along with the different users to access the data transactions. The cloud-based database transaction used for the benchmarks of the standard Database. The transaction processing performance provides the durability of guaranteeing design considerations with the efficient way of the front-end side and back-end side of database transactions. The production of scale-out storage systems, the design of a communication subsystem plays a vital role in determining the overall performance of these systems. Load balancing deployment algorithm across the workload amounts, maximize the activity of overlapping and provides multiple connections between storage servers to maximize transaction purpose along with the clients and servers [1]. The significant performance of larger memory and multi-core applications establish the resident systems across the improved throughput and query latency of data transactions. Their performance no longer limited by the efficiency and scalability over the indexing, logging, transaction compilation, and concurrency control [2].

The database management systems (DBMSs) as a service user through the cloud environment. The high-speed networks evolve the server-less alternative workload fluctuations from physical storage that yield a low performance, high availability to users, and reliability through the database instances of guaranteeing the total cost of ownership optimization. The design can robustly estimate the substantial effort that leads to the single point of the database instance and storage managing both transaction log and data pages. The front-end side and efficient data durability in the back-end side establish the design considerations for the performance of transaction processing among the clients and servers within the benchmarks of better-optimized performance [3]. The storage capacity and the limited processing distributed among the server and the different users, which is addressed by cloud computing. The production of data processing, data storage, data visualization provides the capability of limitless storage and processing through the cloud DBMS [4]. SaaS (Software as a Service) deployed and enables better data transactions in cloud infrastructure. Load refers to the CPU load, network load, and the memory capacity of every server. This load could manage by using different load balancing techniques. The objective of load balancing algorithms is to make sure that every server of the cloud is busy processing a sort of operation. The scalability requires the cloud service provider and end-users with the computational resources that prevent users from utilizing logless transactions. The communication channels and the various computational resources originate from the multitenant systems along with the assignment of a trustworthy cloud service provider. However, the cloud service provider allows maximum data transactions with the irrespective of the behaviors of timeout error tolerance to maximize resource utilization from various frameworks. The significant performance improvement in terms of latency and throughput over the different clients and servers[5].

The read or write transaction of the same data item estimates the more than one transaction, which called concurrent transactions. The concurrent transactions will lead to an inconsistent

database that enables the simultaneous execution of such efficient transactions to lead to better read-write transactions in a cloud DBMS. The factors are following as

- ❖ Replication of data
- ❖ Consistency
- ❖ Concurrency of data items
- ❖ Load Balancing
- ❖ Locking Mechanism
- ❖ Read Write data transactions

Objective

- In this part, the database transactions established the read and write transactions across the client, and the server makes the increased response time.
- For read and write transactions, the load balancing, and locking mechanism enable the database transactions with the reduced response time transaction.

Paper Organization:

- For database transactions and data storage functions in a cloud environment, the Highly Available Commit (HACommit) architecture used to make the read and write transactions across the logless commit process. Its performance to be discussed in Section IV.
- The performance analysis of the HACommit architecture and its read and write transaction shown in Section V.

II. LITERATURE SURVEY

[6] proposed the suitable frameworks that offer Cloud computing for the clustering of distributed databases effectively for the data storage and replacement of read-write data transactions. The Cloud framework provides a lack of misunderstanding to the process of addition and the service integration with holistic approaches that enable the data users to guides the transactions of database applications and the data storage. Cloud-based technology provides different results over the different levels of database transactions to apply and the provision based user applications such as query optimization, data-related service level objective management, robust data analytics, and data privacy for the increased application quality. [7] stated the effective solution for concurrency and to solve the database problems. The database systems and its effective transactions conflict the negative effects on the performance of the system. The positive solution for effective concurrency can considered, which applied among the different users [8] proposed an improved two-phase locking provides the transparency control policies that minimize the transactions conflict and achieve the deadlock-free locking. The replication mechanisms used to prevent the losses in databases and give high availability across persistent memory failures. NVRAM replication estimates the optimized overhead replication through low latency and minimal throughput.

[9] stated the highly-available data stores provide the distribution of data transaction can take more processing time, such as bisection of the overall transaction in the commit based distribution. The database transaction provides the write-ahead logging (WAL) approaches to enable the roundtrip interaction in the commit distribution. [10] stated the stable storage device used to secure the data loss due to the failure of the device, the database system of a primary server replicates periodically, and the backup servers by copying log entries over networks. The modern database system enables the transaction execution, which is highly parallelized, the centralized logging through the single I/O channel that leads to inhibit the scalability of database transactions. [11] proposed the distributed database transactions provides the reliability among the reflection of commit protocols. The commitment protocols ensure that whether all the changes in a transaction are applied or not. The efficient commitment process ensures through the database community, which used for two-phase commit protocol along with the efficient transaction processing systems. [12] proposed the identification of suitable data management platforms to store and query this data is necessary. Despite its popularity and efficiency in processing various types of big data, there is no single-guided study of how NoSQL data stores will behave with the Internet of Things (IoT) datasets along with DBaaS and the Number of unstructured data storage functions. [13] proposed the context of cloud computing provides query optimization among the industrial commodities. The query optimization enables the taxonomy of improved demand for query performance among the clients and servers. [14] proposed the large data centers hosting Cloud Computing servers through the significant progression to handle the distributed edge computing demand as well as security attack and cloud software security requirements.

[15] stated the combination of Cloud computing, the Internet of Things (IoT), and big data analytics technologies provides the association of eliminates the stressful wait times and overcoming geographical stress. This paper focuses on Tele-Rehabilitation as a Service (TRaaS). The big healthcare data generates the remote rehabilitation components used by the patients that required for the hospital Cloud applications. [16] proposed the modern NoSQL systems have the storage layer that enumerates the database community and the operating systems community for storage purposes in LSM-trees for the reason of general taxonomy of their strengths and trade-offs of NoSQL applications. [17] stated the Computerization, digitalization, and the Internet through the interconnectivity of the Software, store data, and provide the hardware applications. [18] stated the Oracle Cloud provides the Platform as a service through the availability of fully managed ORACLE databases to perform the oracle cloud account. The system established the self-provisioning of hardware resources, elasticity and scalability, and the implications of delivery as a service and the characteristics of cloud computing and the cloud DBMS systems.

[19] stated the Cloud data centers enable and satisfy the provider profit and data Replication Strategy along with the execution time, data replication, Response Time, threshold value, and the read-write data transactions with the consistency scheme of control policies among the database transactions. [20] proposed the extraction, transformation, and load (ETL) used to develop the strategic plan and a communication flow among the clients and the servers. The data

integration involves the context of data, client-server performance, limited sources to the databases, and the efficient data storage across the multiple database transactions in the presence of cloud DBMS framework.

III. PROBLEM STATEMENT

The cloud database transaction is an essential factor for data storage and its related function. The distribution of database transactions may lead to challenges like data security, performance, data privacy, resource utilization, and high availability functions with the high quality of data storage in cloud DBMS. The inefficient ways of database transactions reflect the read and write functions across the cloud infrastructure. The response times of the system also affect the read and write capacity. Due to the database transactions, the replication of data makes the inconsistency of database transaction problems. Due to the read and write transactions, the response time creates the conflict of data storage and database transactions. With the corporate adoption of cloud computing, the significant increase of cloud options established the database services in the form of cloud databases or Database-as-a-service (DaaS). These early applications put a priority on read access because the ratio of reads to writes was very high. Delivering high-Performance read access was the primary purchase criteria. However, this is changing. Adding another database server is as simple as splitting the data across one more server. Many user requests involve related information.

DFCL (Deadlock Free Cell Lock)

The existing method DFCL algorithm detriments the overhead of concurrent data transactions. DFCL, referred to as Deadlock Free Cell Lock, depends on the locking system applied to the low degree of locking. To access the same data concurrently and enables a small percentage of conflict through the many transactions. The algorithm proceeds the operations on its required cells, which is depends on the lock mode. DFCL provides either read or write transactions simultaneously within the cells. It also eliminates the deadlock issues while locking the cells by forcing the waiting transaction to pass into the rollback or the commit phase. Our proposed algorithm improves the performance of the Database and the transactions. During concurrency transactions, the transactions may affect due to the conflict and deadlock condition of data storage. The deadlock issues generated from an infinite waiting time of transactions for data lock condition. During data transactions, DFCL detriments in the high overhead of concurrent data transactions with the Number of increased waiting times.

To overcome the issues, to utilize the HACommit architecture for active transactions along with the requirement of less time of database transactions for the orientation of business applications. Hence the HACommit architecture improves the performance of the cloud database transactions concurrent transparency, enhanced read to write data transactions.

IV.PROPOSED METHODOLOGY

A Cloud DBMS is a distributed database that brings computing as a service instead of a product. It shares the resources, information, and Software between multiple devices over the Internet. Nowadays, it is growing significantly. As a result, database management tasks outsourced to third parties, just like putting it into the cloud for a much lower cost. The structure of the cloud computing database and its functioning in collaboration with other nodes observed under the Database as a service. Many e-commerce companies are getting benefits from DB as a service. For useful read and write transactions, the HACommit architecture enables the effective, consistent, and replication of data transactions among the clients and servers. The HACommit provides the non-blocking mechanism under the client and server failure along with the nodes of data transactions.

HACommit architecture (Highly Available Commit) designed for highly-available data transactions. The highly available data transactions established the one phase-based read and write data transactions. The partition of data to be distributed through the network of different servers to attains the improved scalability, reliability of the server. The high availability of data used for the replication over the various servers. The data replication and distribution are except for underlying the multiple data transactions. The use of front-end servers provides the clients. The client starts a data transaction. The server can hold the data transaction along with the use of participants commit nodes like A, B, C, D while servers are holding replicas by the presence of the HACommit that provides the client and server such as a transaction processing based client-side library. The Highly Available Commit (HACommit) data transaction enables the improved concurrency of data transactions without waiting time delays.

The read (Read Log Equalizer) and write (Write Log Equalizer) transactions done over the process when the HACommit established the job assignment for each participant commit nodes, the lock-based ACK, and data transactions through the client-server interfaces. The commit interface enables the client based processing; the last Operation processing featured as a normal commit process. The client or participant failures established the recovery process.

The data transaction starts an execution phase in the HACommit. The transactions perform the read and write transactions with the inconsistent replication of solutions, except for the last transaction. For the last Operation, the client displays to all the participant nodes like A, B, C, D, which is the last transaction processing operation. Overall the nodes check to vote YES or NO decision for the commit data transaction process, which depends upon the consistency of data transactions, data integrity, and concurrency control like a lock-based mechanism. The participant nodes replication enables the context information transaction before the response to the client. The client receives the commit like ACK from all the participant nodes as well as the last processing operation. Now the commit process starts.

Now the client picks the decision-making of whether to commit or abort when the concurrent data transactions occurred. So the client makes to select the commit only the transaction if all participant commit nodes vote YES like as one by one. Now the client decides for data

transaction reads to the participant's nodes and their replicated commit nodes. Then the client ends the data transactions. Once the client received the acknowledgment from the replicated commit nodes, the replication of data item to be read until all the other replicated commit nodes locked by applying the locking mechanism approach. Despite the failures occurred due to the error in the replication of the participant commit nodes in the existing methods that to be recovered and solved along with the Number of increased commit operation within the commit nodes.

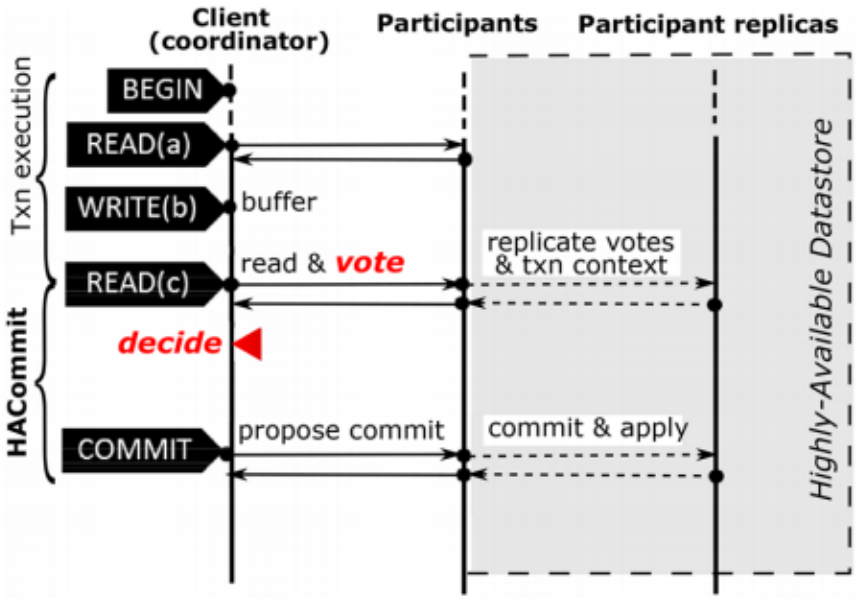

Figure 1. HACommit based One-Phase Transactions

HACommit provides the one phase logless data transactions such as without deadlock or waits time of participant nodes. The execution of data transactions enables the level of locking among the nodes during a read-write data transaction. In reasonable condition, the read-write transaction done through the ACK sends and then received. The proposed HACommit attains the specific feature i.e., decide and vote approach. When the transaction starts, the participant nodes operate through the commit phase. When the transaction to be committed, the process never faces the wait and delay issues. The number of transactions increased and also without inconsistent data transactions enabled. When the read transactions completed by the presence of a locking approach, vote to commit Yes. Then the write data transactions done and also the lock to be released and without any difficulty of data transactions. The locking approach never makes the complexity related issues. The Operation enhanced the improvement of concurrency as well as eliminates the wait time of data transactions.

HACommit established the process like the Consensus algorithm for the actual less Number of client-server failures while during the data transactions. HACommit also provides the

specification like replacing or eliminate the client and server failure. When the client-server failure may occur, the transaction of the whole process leads to failure, execution time to be high, etc.

4.1 Consensus Algorithm

Consensus Algorithm is the process of accepting the one database transactions among the all participant nodes without any failures of client and server failures. The acceptance of database transactions depends on the consensus value i.e., the minimum number of active connections among the capacity of the server. This algorithm provides the vote-before-decide among the commit transaction, which is lesser than the traditional commit issues. Consensus algorithm used while data transactions without delay and no wait time of concurrent data transactions. The client decides the decision among the nodes. The memory capacity of the server depends on the consensus value. To regulate the consensus values among the participant nodes. The consensus value depends upon the memory capacity of the participant commit nodes and the applied load across the commit nodes. The problem may arise when the difficulties in the participant nodes or their respective communication among the client and server, the overall transaction experienced as processing delay.

Consensus value, C = Active connections of the Workloads – Memory Capacity

The Consensus algorithm experienced as high durability when the replication of a database transaction occurs. While the last transactions perform, the client sends a message to the participants to commit nodes that hold the data. The client sends the message to the other commit nodes that provide to have no processing of transactions, i.e., empty Operation performs. The last Operation acts as a read, a write, or a meaningless operation—the ACID property, whether a commit checks whether for the participants of commit nodes violation. After the last Operation, the participant vote YES, or else Vote to NO to restart the data transactions or not.

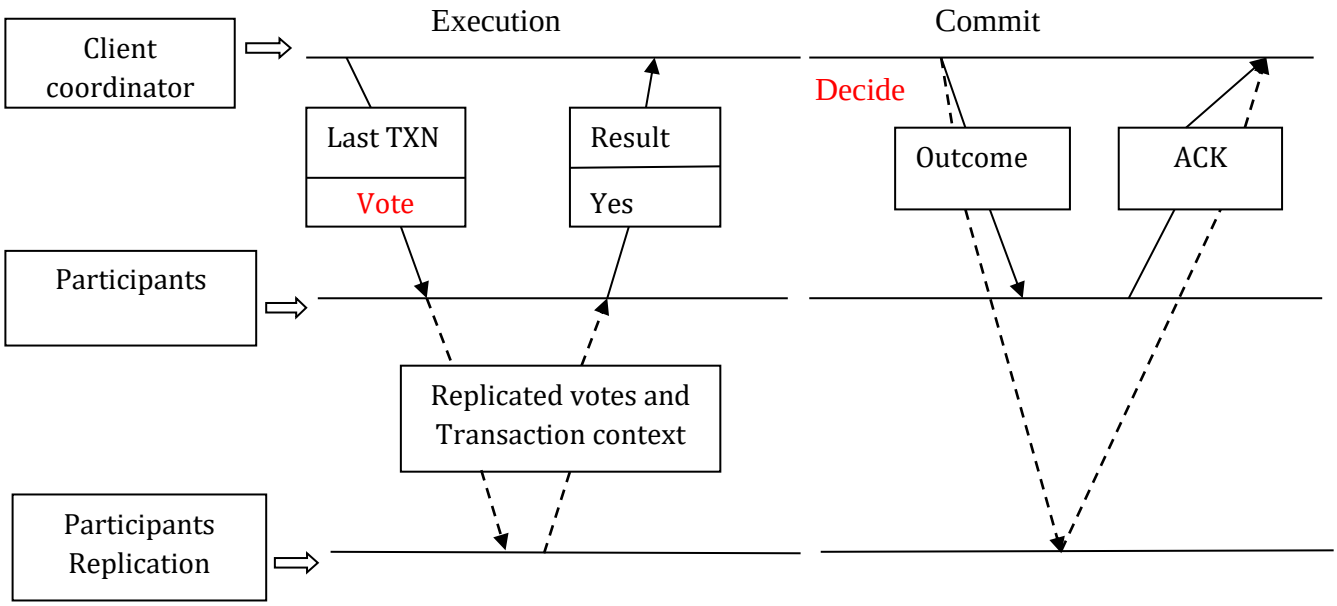

Figure 2. Consensus algorithm based client coordinator

Figure 2 shows the Consensus algorithm based client coordinator.

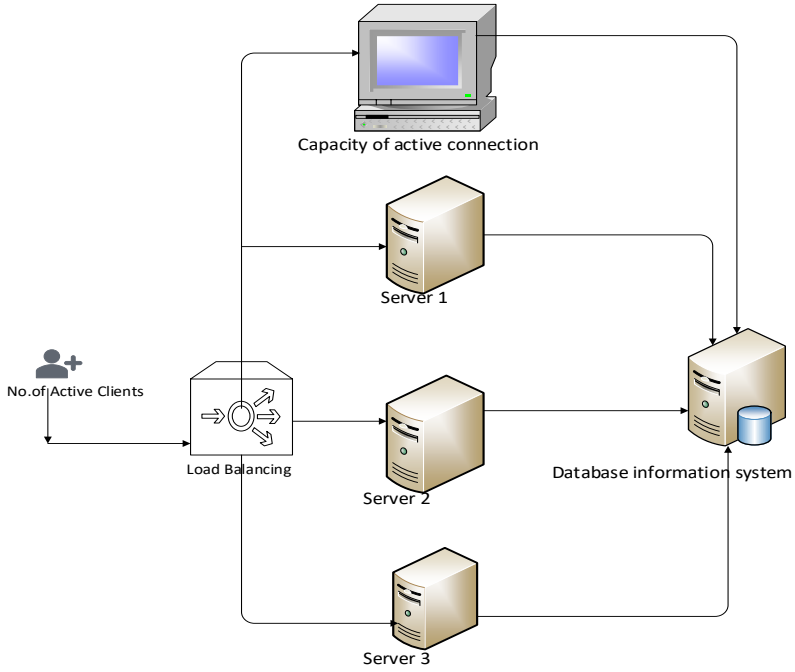

Figure 3 Read and Write Log Equalizer

4.2 Read and Write Log Equalizer (RLE and WLE)

The read the data transaction done with the presence of load balancer based concurrency control policy. The read transaction named as commit mode. i.e., HACommit initially decides whether to vote or not. HACommit provides the decision like the vote to YES means; the other commit nodes locked with the conscious of other participant nodes and its active connections. The read transactions have highly available data stores among the different commit nodes. Then read the transaction done, the other commit nodes to be released. The client coordinator receives the read data transactions without any wait time, no delay, and as well as consistent data transactions. The capacity of the active connections established the load balancing along with the client-server database implementation systems. The transaction processing to solve inconsistency and to control concurrency using lock managers. A dynamic load balancer distributes the load among servers. Hence it needs to satisfy the user with minimum response time and correct data.

Hence the load balancing approach used for the proper utilization of resources to improve the performance of the overall system. It also maximizes throughput. For distributing the read and write load among servers based on their capacity and minimum active connections. As the client requests are distributed based on their capacity, servers can provide a quick response. And also, the minimum active connection server is considered for service. So, it can provide better service to the clients with minimum response time and maximum throughput.

The write transaction done through the consistent replication of database transaction. The write transaction that also includes the transaction context. The applicable write data transaction provides the replication participant commit nodes. The participant commit node failures required for the consistent write data transactions. The write requests to the participant commit nodes sent by the client. To choose to write transactions along with the conscious execution of write transactions. The transaction context distributes among the commit nodes. The overall write data transactions manipulate through the stable storage, before the changes write to the database transactions. The change in the read or write attains the equal data transactions. Several works belong to data replication and high scalability of data transactions in the cloud environment, which takes no waiting to write transactions. All the modifications write to a log before the data transactions done. It enables the processing time. The time taken for read to write transactions to be quiet high. Because of the write transaction ensembles, the participant commit node failures. During transaction depends on the client and server failures. The commit node replication does not evolve the inconsistent data transactions. So the participant commit nodes never process the same write transactions. The replication of data transaction provides the relevant write transaction to the server for active write database transactions. Consensus algorithm used to detect the client and server failures, then to write the data transaction along with the lock released concept to the same data at different times, such as increased write time compared to the read transaction time. The Number of servers and the increased Number of job assignment specifies the response time as well as the latency time during write database transactions.

4.4 Algorithm 1: A read and write data transaction

// the transaction among the A, B, C, D commit nodes
start the transaction;
for all n − 1 transactions do
send the transaction to the participant commit nodes and the results during processing ;
end
// data transaction like read-write transaction for one cycle
send the ACK to the participants commit nodes
assign the job and load balancing to all the commit nodes
receives ACK
Lock the other B, C, D commit nodes
Performs the read transaction
If successful read transactions
Such as consensus value = active connections – memory capacity

release the locks
performs the write transactions
end
// process the multiple read-write transaction
Performs multiple read write transactions
end
// process the last transaction
send the n-th transaction to the participants commit nodes;
send relevant write transaction to and incurs the votes from all participant nodes;
waits for the results from all commit node participants;
end the transaction safely;
// process the commit starts
For all commit nodes participants do
set result = Commit read and write transaction;
send message as client failure or server failure;
timeout the transaction;
end
wait until the message from all the commit nodes participants;
finally not receiving message;
// then
end the transaction;

4.5 Fault Tolerance

Fault-tolerant used for the active process of determining client failure or server failures during the data transactions. If the failure of nodes occurred, the connection to be a timeout. The fault-tolerant used to identify when and where. The failures are not common during the failed replication process. It will detect and replaced quickly with the normal state of the client and server. HACommit enables the non-blocking process under the client-server failure with the participant to commit nodes of data transactions. When fewer than a quorum of participants fails transactions.

V.RESULTS AND DISCUSSION

The HACommit architecture based read-write transaction simulated using the CloudSIM tool. The workload generation for replication of consistent data items to provide the read and write transactions. The parameters involve classifying the write database transactions with the corresponding data centers and its response time calculations. The Read Log Equalizer and Write Log Equalizer established the Number of replicated nodes, read transactions, write transactions, and response time due to the database transactions. The load variation balanced the write database transactions to enables the speed of the write function. The ratio of load variation and its rate estimated through the corresponding Number of replicated nodes. The performance of results compared with the existing work methodologies. From the results, the existing method and its approaches to the DFCL algorithm compared with the proposed HACommit architecture for a highly compelling read and write data transactions.

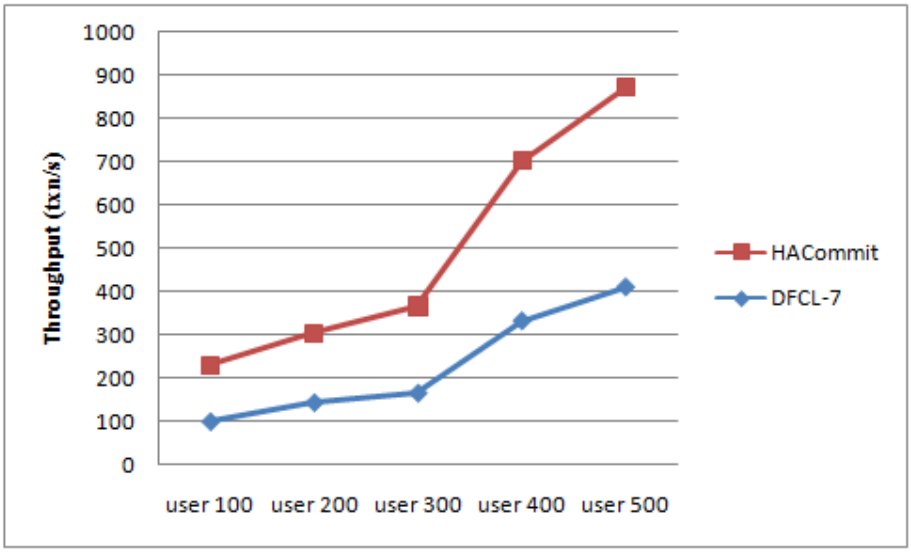

Figure 4 Number of Users Vs. Throughput

Figure 4 Number of users Vs. Throughput. The load factor depends upon the speed of the transactions during the resource allocation of all other transaction nodes. The resource consumption of all other nodes enables the request during the load instant. The different load condition allows when the same service request and the speedup time may increase, based on the difference between the current load and the load factors of the transaction nodes. The load enumerates the current CPU usage, memory usage, and network of the transmission nodes. The experimental result shows the Number of replicated nodes. When there is a change in the resource status, the speedup time increases in linear order. For read transactions, the Number of replicated nodes and its read transaction variation.

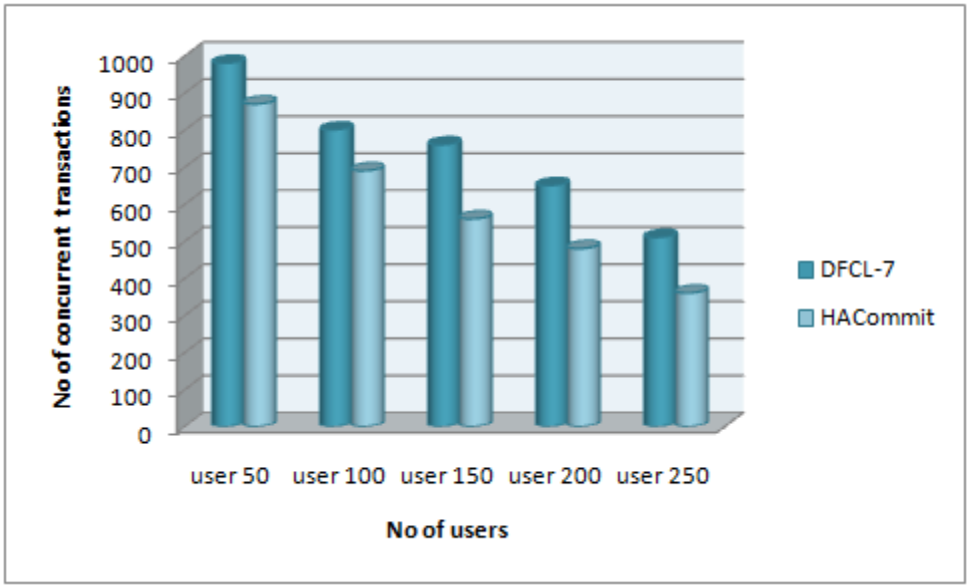

Figure 5 Number of users Vs. Number of concurrent transactions

Figure 5 represents the Number of users Vs. Number of concurrent transactions.

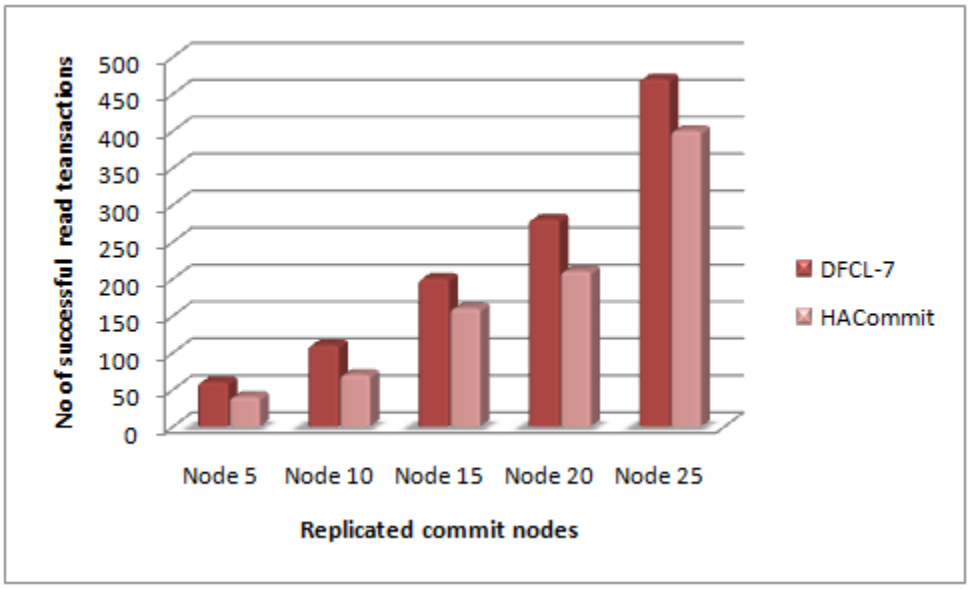

Figure 6. No of replicated commit nodes Vs. No of successful read transactions

Figure 6 illustrates the Number of replicated commit nodes Vs. No successful read transactions. The Number of successful read transactions. HACommit based read and write transaction displays the function during read phase without holding any locks. RLE and WLE maintain consistency in the validation phase and provide the write phase with the updating of the database transactions.

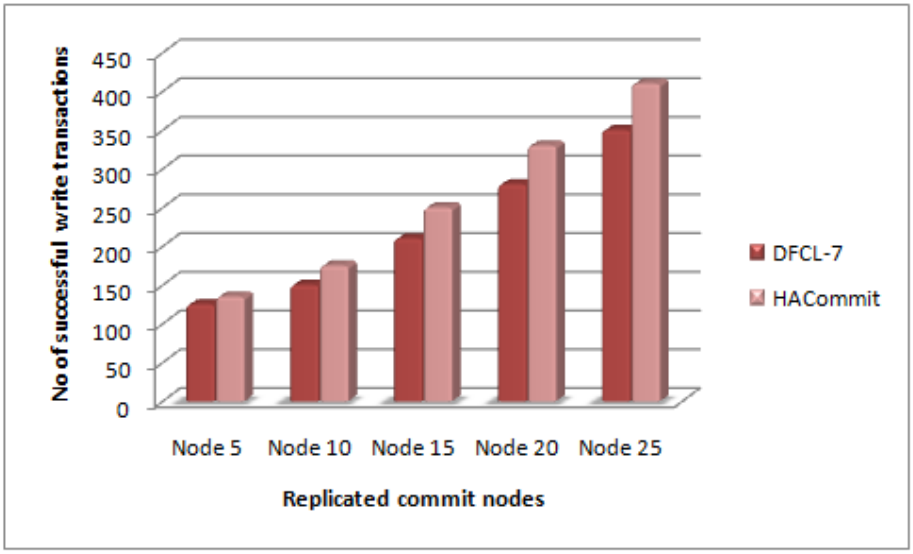

Figure 7 Number of replicated commit nodes Vs. The Number of successful write transactions.

Figure 7 represents the Number of replicated commit nodes Vs. The Number of successful write transactions. The HACommit provides the response time and the waiting time across the concurrent transactions along with the number of successful transactions of read and write database transactions. While increases in response time and reduces the waiting time for concurrent transactions. The execution of concurrent transactions that ensures the serializability of the transactions.

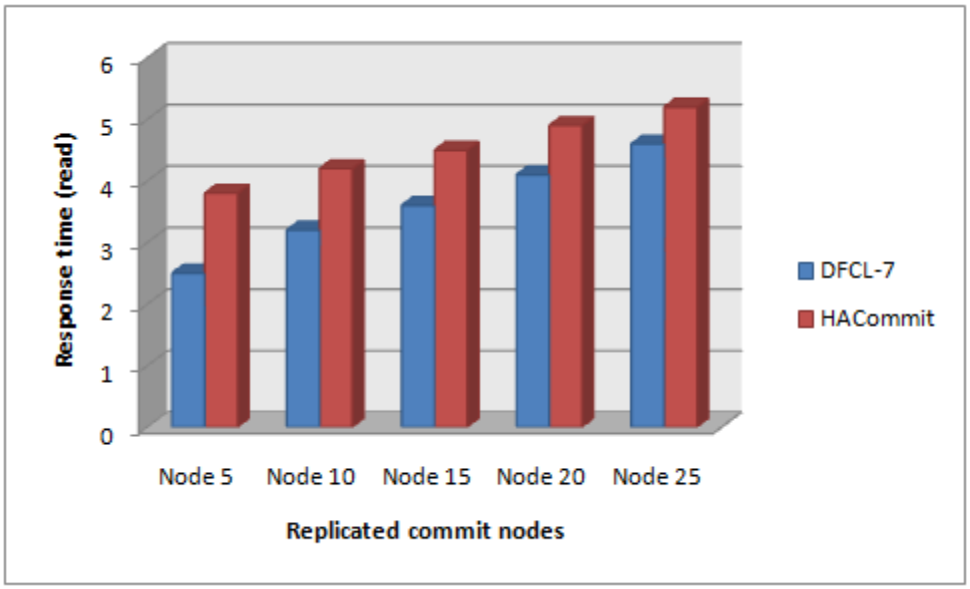

Figure 8 No of replicated commit nodes Vs. Response time (Read)

Figure 8 represents the Number of replicated commit nodes Vs. Response time (Read)

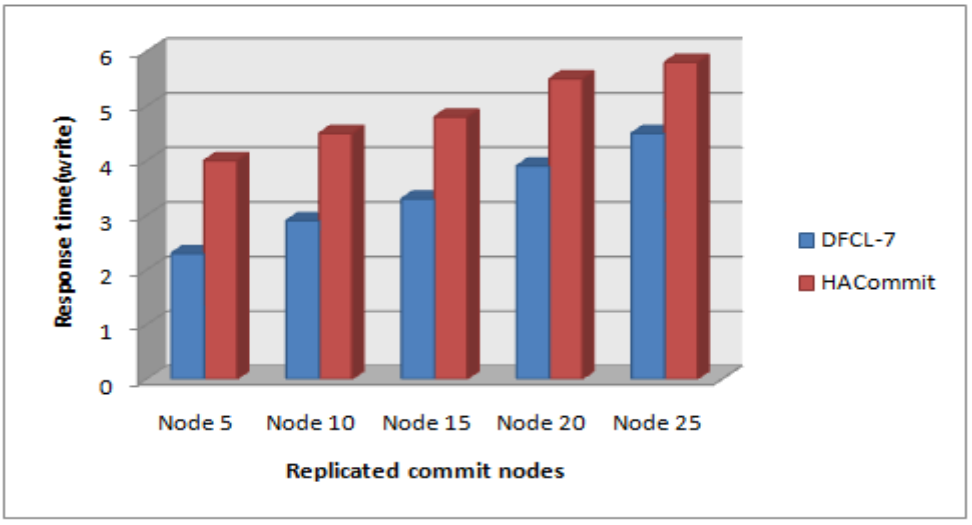

Figure 9 No of replicated commit nodes Vs. Response time (write)

Figure 9 represents the Number of replicated commit nodes Vs. Response time (write). Figure 10 illustrates the Operation per transaction Vs. Average latency.

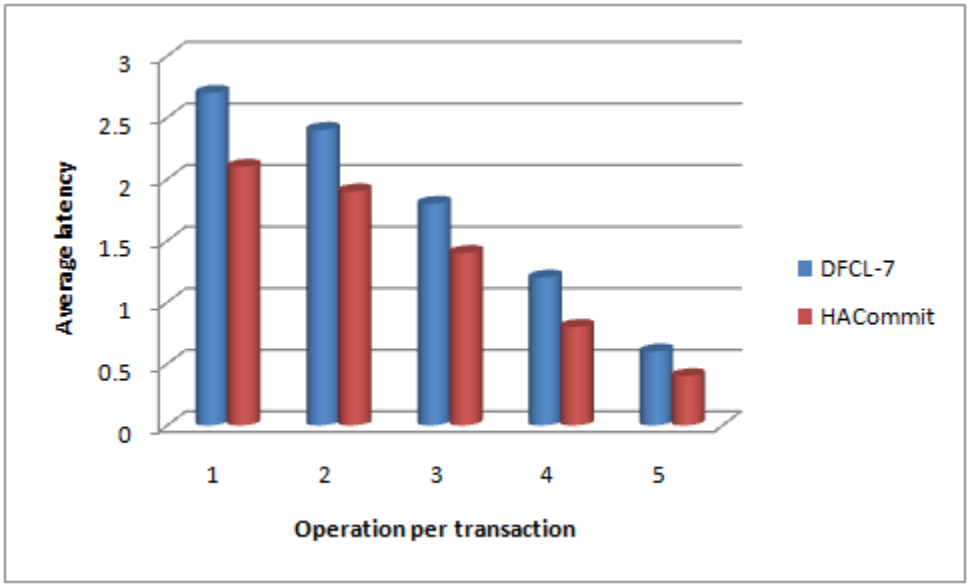

Figure 10 Operation per transaction Vs. Average latency

CONCLUSION

Cloud DBMS based data transactions are effective through the balanced distribution of workloads among the replicated commit nodes by the presence of HACommit. The HACommit architecture based database transactions provides the logless one-phase commit process along with the highly available, reliable, and scalable for business applications. HACommit kept the approach like as vote-before-decide through processing distribution of the last processing transaction. The voting process overlapped by the last operation processing to the transaction of the commit process in a single-phase commit process. The aim of read and write log equalizer is to satisfy the user while distributing load among the several commit nodes or servers. The Consensus configuration information presence of transaction context structure that used to identify the client failures or not. The HACommit process effective read transactions and write transactions. When read transactions occur, the time to be reduced and during write transactions occurs; the taken time is high by comparing the read transactions while selects the job assignment for each replicated commit node of the transactions.

References

[1] Dr.K.Chitra, A.Aasha Begum, Capacity based Load Balancing for Handling Read only Transactions in Cloud Data Storage, International Journal of Advanced Science and Technology, ISSN : 2005-4238, vol.29, Iss-2, 4412-4419, 2020. http://sersc.org/journals/index.php/IJAST/article/view/5330.

[2] A.Aasha Begum, Dr.K.Chitra, A Survey on Cloud DBMS Enabled Data Transactions and Data Storage, International Journal of Emerging Technologies and Innovative Research (www.jetir.org), ISSN:2349-5162, Vol.7, Issue 3, page no.21-29, March-2020, Available :http://www.jetir.org/papers/JETIRDP06006.pdf.

[3] Dr.K.Chitra, A.Aasha Begum, Database Transaction Processing with Effective Locking Mechanism for Consistency in Cloud Database, International Journal of Computer Sciences and Engineering, Vol.6 , Issue 11, Page no. 1006-1009, Nov-2018.

[4] A.Aasha Begum, Dr.K.Chitra, Formation of Single and Multinode Clusters in Hadoop Distributed File System, World Congress on Computing and Communication Technologies (WCCCT), 162-164, 2017.

[5] A.Aasha Begum, Dr.K.Chitra, Fully Automated Auto Scale Data Processing with Cloud Dataflow, Fifth International Conference On Design And Applications Of Structures, Drives, Communicational and Computing Systems (ICDASDC 2016) (www.klncit.edu.in), ISBN :978-1-63535-335-8, page no.444-447, Feb-2016.

[6] A.Aasha Begum, Dr.K.Chitra, A study on Mobile Cloud Computing Services and its Advantages, International Journal of Emerging Trends in Computing Technologies, MASIVJ Bi-Annual, ISSN:2454-4558, vol.1, Special Issue-4, Page no.1-7, Feb 2016.

[7] U. Song, B. Jeong, S. Park, and K. Lee, "Optimizing communication performance in the scale-out storage system," *Cluster Computing,* vol. 22, pp. 335-346, 2019.

[8] H. Hu, X. Zhou, T. Zhu, W. Qian, and A. Zhou, "In-memory transaction processing: efficiency and scalability considerations," *Knowledge and Information Systems,* pp. 1-32, 2019.

[9] W. Sul, H. Y. Yeom, and H. Jung, "Towards Sustainable High-Performance Transaction Processing in Cloud-based DBMS," *Cluster Computing,* vol. 22, pp. 135-145, 2019.

[10] F. Firouzi and B. Farahani, "Architecting IoT Cloud," in *Intelligent Internet of Things*, ed: Springer, 2020, pp. 173-241.

[11] G. Pallavi and P. Jayarekha, "Secure and efficient multitenant database management system for cloud computing environment," *International Journal of Information Technology,* pp. 1-9, 2020.

[12] S. Mazumdar, D. Seybold, K. Kritikos, and Y. Verginadis, "A survey on data storage and placement methodologies for cloud-big data ecosystem," *Journal of Big Data,* vol. 6, p. 15, 2019.

[13] M. Mohamed, M. Badawy, and E.-S. Ayman, "An improved algorithm for database concurrency control," *International Journal of Information Technology,* vol. 11, pp. 21-30, 2019.

[14] M. Zarubin, T. Kissinger, D. Habich, T. Willhalm, and W. Lehner, "Efficient compute node-local replication mechanisms for NVRAM-centric data structures," *The VLDB Journal,* pp. 1-21, 2019.

[15] Y. Zhu, S. Y. Philip, G. Yi, M. Guo, W. Ma, J. Liu*, et al.*, "Logless one-phase commit made possible for highly-available datastores," *Distributed and Parallel Databases,* vol. 38, pp. 101-126, 2020.

[16] H. Zhou, J. Guo, H. Hu, W. Qian, X. Zhou, and A. Zhou, "Plover: parallel logging for replication systems," *Frontiers of Computer Science,* vol. 14, p. 144606, 2020.

[17] S. Gupta and M. Sadoghi, "Efficient and non-blocking agreement protocols," *Distributed and Parallel Databases,* pp. 1-47, 2019.

[18] A. Hendawi, J. Gupta, J. Liu, A. Teredesai, N. Ramakrishnan, M. Shah*, et al.*, "Benchmarking large-scale data management for Internet of Things," *The Journal of Supercomputing,* vol. 75, pp. 8207-8230, 2019.

[19] A. Sebaa and A. Tari, "Query optimization in cloud environments: challenges, taxonomy, and techniques," *The Journal of Supercomputing,* vol. 75, pp. 5420-5450, 2019.

[20] N. K. Sehgal, P. C. P. Bhatt, and J. M. Acken, "Foundations of Cloud Computing and Information Security," in *Cloud Computing with Security*, ed: Springer, 2020, pp. 13-48.

[21] A. Celesti, A. Lay-Ekuakille, J. Wan, M. Fazio, F. Celesti, A. Romano*, et al.*, "Information management in IoT cloud-based tele-rehabilitation as a service for smart cities: Comparison of NoSQL approaches," *Measurement,* vol. 151, p. 107218, 2020.

[22] C. Luo and M. J. Carey, "LSM-based storage techniques: a survey," *The VLDB Journal,* vol. 29, pp. 393-418, 2020.

[23] M. T. Jakóbczyk, "Introducing Oracle Cloud Infrastructure," in *Practical Oracle Cloud Infrastructure*, ed: Springer, 2020, pp. 1-48.

[24] M. T. Jakóbczyk, "Autonomous Database," in *Practical Oracle Cloud Infrastructure*, ed: Springer, 2020, pp. 347-408.

[25] R. Mokadem and A. Hameurlain, "A data replication strategy with tenant performance and provider economic profit guarantees in Cloud data centers," *Journal of Systems and Software,* vol. 159, p. 110447, 2020.

[26] J. Goldfedder, "Choosing an ETL Tool," in *Building a Data Integration Team*, ed: Springer, 2020, pp. 75-101.