Re-engineering Grid-Based Quorum Replication into Binary Vote Assignment on Cloud: A Scalable Approach for Strong Consistency in Cloud Databases

Ainul Azila Che Fauzi[1], Noor Ashafiqa[2], Asiah Mat[3], Syerina Azlin Md Nasir[4], A. Noraziah[5]
Faculty of Computer and Mathematical Sciences, Universiti Teknologi MARA Cawangan Kelantan, Machang, Kelantan [1, 2, 3, 4]
Faculty of Computing, Universiti Malaysia Pahang Al-Sultan Abdullah, Pahang, Malaysia[5]

***Abstract*—The growth of cloud computing has heightened the demand for replication strategies that ensure strong consistency, high availability, and low communication cost across distributed infrastructures. Existing systems such as DynamoDB, FoundationDB, and GeoGauss illustrate different design trade-offs but face limitations in balancing latency, correctness, and resilience under dynamic workloads. This study proposes the Binary Vote Assignment in Cloud (BVAC), a cloud-native replication algorithm re-engineered from the Binary Vote Assignment on Grid Quorum (BVAGQ). BVAC organizes replicas in a logical grid structure and employs binary voting weights with a Commit Coordination (BCC) mechanism to enforce quorum-validated commits, representing a form of quorum-based replication. This design maintains serializable consistency, minimizes replication conflicts, and reduces low communication cost through fixed-size quorums of three to five replicas. Experimental results demonstrate that BVAC sustains fault tolerance, achieves cloud database replication efficiency, and sustains high data availability via multiple valid quorum paths. By avoiding the heavy coordination cost and infrastructure footprint of current systems, BVAC provides a scalable and cost-efficient replication strategy tailored for modern cloud workloads. The study establishes BVAC as an advancement in distributed data management and a foundation for future adaptive and multi-cloud replication frameworks.**

Keywords*—Binary Vote Assignment in Cloud (BVAC); cloud database replication; fault tolerance; high availability; quorum-based replication; strong consistency

I. Introduction

Distributed computing environments present significant challenges, particularly in maintaining high availability and reliability of data. Data replication has long been recognized as a fundamental technique to address these challenges, as it enables systems to provide fault tolerance, load balancing, and continuous access to critical information [1, 2, 3, 4].

In traditional grid computing environments, data replication was introduced to enhance availability and dependability by distributing replicas across multiple grid nodes. This approach allowed systems to tolerate node failures, support concurrent access, and reduce the risk of data loss in large-scale scientific and computational applications [5]. Techniques such as static replication and dynamic replication policies were widely deployed to ensure that frequently accessed data remained available across geographically dispersed resources [6].

While grid-based replication strategies achieved significant success in improving fault tolerance and data accessibility, they also exhibited several limitations. Grid environments are inherently tightly coupled and resource-constrained, which often restricts scalability and adaptability. Furthermore, replication overhead was difficult to manage, and consistency across replicas was not always guaranteed due to network heterogeneity and limited elasticity of grid resources [7]. These challenges made grid-based replication less suitable for modern workloads that demand real-time responsiveness and global accessibility.

Several replication strategies have been proposed to address the challenges of managing data in cloud environments. Broadly, these methods can be categorized into synchronous replication and asynchronous replication. Synchronous replication ensures strong consistency by updating all replicas simultaneously, which is suitable for mission-critical applications but often introduces high latency, especially in geographically distributed clouds [5]. On the other hand, asynchronous replication reduces latency by allowing updates to propagate to replicas at a later stage, thereby improving performance but at the cost of potential data inconsistency during failures [6].

In addition to these traditional methods, several hybrid approaches have been developed to strike a balance between consistency and performance. For example, techniques such as quorum-based replication and majority consensus protocols allow systems to achieve partial synchrony, ensuring data reliability without incurring the full latency penalty of synchronous models [7]. Multi-cloud replication frameworks have also been explored to enhance disaster recovery and reduce the risk of vendor lock-in, but these often come with additional management complexity and resource overhead [8, 9].

Despite these advances, current replication approaches remain limited in handling the highly elastic and dynamic nature of modern cloud workloads [2, 10]. In practice, cloud environments introduce additional challenges such as multi-tenancy, where multiple independent users share the same infrastructure, often leading to unpredictable performance interference. Moreover, geo distribution of cloud datacenters creates latency asymmetries and raises the difficulty of maintaining consistency across regions. Cost-efficiency also becomes a critical factor, as replication policies directly affect storage, bandwidth, and operational costs in pay-as-you-go

models. Traditional replication mechanisms often fail to adapt to these requirements, as they were not originally designed to optimize for resource elasticity, cost sensitivity, and tenant isolation. This gap underscores the importance of designing new replication strategies that can simultaneously ensure high availability, strong consistency, and performance while remaining cost-aware and adaptable to real-time workload dynamics.

With the rise of cloud computing, replication challenges have become more complex. Cloud infrastructures demand solutions that can handle multi-tenant environments and flexible resource allocation while maintaining low latency and strong consistency [7, 10]. Existing replication mechanisms do not adequately address these requirements, leaving a gap for innovative approaches that balance consistency, availability, and performance in large-scale, globally distributed cloud systems.

To address these challenges, this study proposes the Binary Vote Assignment on Cloud (BVAC) algorithm, by re-engineering BVAGQ [11] for cloud environments. BVAC is designed to improve data consistency and availability by leveraging a binary voting mechanism in quorum-based replication, where replica servers are assigned voting weights to determine the validity of transactions. This voting scheme minimizes replication conflicts, enhances fault tolerance, and reduces the communication overhead typically associated with quorum-based replication. Unlike previous grid implementations, the proposed BVAC adapts its design to the elasticity and scalability of cloud infrastructures, enabling efficient replica placement across geographically distributed nodes. By re-engineering BVAGQ for cloud environments, this research aims to bridge the gap between traditional quorum-based replication in grids and the dynamic requirements of modern cloud systems. Unlike existing approaches, the proposed BVAC explicitly addresses multi-tenancy, geo-distributed deployments, and cost-efficiency by incorporating a lightweight voting mechanism that adapts to workload variability while minimizing communication overhead. This makes BVAC particularly well-suited for cloud infrastructures where scalability, elasticity, and operational costs are as critical as data consistency and availability.

The remainder of this paper is organized as follows. Section II presents the literature review on replication strategies in grid and cloud environments. Section III describes the design of the proposed Binary Vote Assignment on Cloud (BVAC) algorithm, including its voting scheme and replica placement strategy. Section IV presents the experimental setup and evaluation methodology, and discusses the results in terms of consistency, availability, and performance. Finally, Section V concludes the paper and outlines directions for future research.

II. Literature Review

Data replication in cloud environments integrates diverse sub-strategies, techniques, and algorithms into cohesive frameworks that sustain system dependability. The replication process is commonly structured around three fundamental phases: identifying frequently accessed data, determining the number of replicas, and selecting their optimal placement. Extensive research has introduced algorithms targeting each of these phases to enhance performance, consistency, and fault tolerance [12, 13]. Modern cloud platforms translate these conceptual strategies into operational practice through protocol-level implementations and system design choices. DynamoDB, for instance, represents a production-grade system in which phase-level replication decisions are directly embedded within concrete protocol mechanisms, thereby exemplifying the alignment of theoretical models with practical deployment.

DynamoDB integrates a tunable-consistency read model into its replication design, providing eventual reads by default for latency reduction and permitting per-request strong reads within a region under ACID semantics with serializable isolation [14, 15]. In multi-region deployments, Global Tables employ asynchronous propagation, which yields eventual cross-region consistency while decoupling regions during inter-region disturbances [15]. The communication path per write remains bounded, as a leader disseminates the update to followers across three (3) availability zones and commits upon a two-of-three quorum, so the critical path corresponds to one intra-region round-trip time with message complexity linear in the replica count, $O(n)$ for $n = 3$. Read placement adheres to the consistency objective, whereby strong reads are directed to the leader and eventual reads may be served by any replica; transactional operations introduce additional coordination rounds via two-phase commit, and any synchronous cross-region configuration, when enabled, elevates both the critical-path round-trip time and the message fan-in. With respect to availability, the three-replica topology tolerates a single replica failure per partition and supports rapid leader re-election, while asynchronously replicated Global Tables allow regional autonomy under wide-area impairments at the expense of temporary divergence; conversely, synchronous multi-region modes trade some write availability to secure stronger cross-region guarantees [15].

Alongside DynamoDB, two further systems illustrate alternative trade-offs, namely FoundationDB and GeoGauss. FoundationDB advances strict serializability across the keyspace through an unbundled control and storage plane comprising commit proxies, resolvers, and log servers, with synchronous replication in a primary region and frequent inclusion of satellite replicas as well as an asynchronous secondary region for disaster recovery [16]. The communication path per writes traverses coordination services and multiple log replicas, often including satellites, which introduces additional hops relative to a three (3) replica quorum and can elevate tail latency under load or under cross-region safety requirements. Steady-state reads obtain a read version from the primary path and avoid extra wide-area round trips. With respect to availability, the architecture tolerates multiple component failures inside a region and supports automatic failover to a secondary region upon primary loss, delivering strong reliability at the expense of additional coordination and replica footprint [16].

GeoGauss pursues strong global consistency for geo-distributed SQL through a full-replica, multi-master architecture that combines epoch-driven optimistic coordination with Raft-style membership, enabling write origination in any region while enforcing a global commit order [17]. The communication cost exceeds single-region designs, since each transaction issues cross-region control traffic for epoch advancement and

validation and ships data to full replicas, which embeds wide-area round-trip time and extra coordination in the commit path, particularly under contention or reconfiguration. Regarding availability, complete replicas across regions furnish resilience and locality for reads and writes, and service can continue through regional failures, while long-haul coordination under partitions or congestion lengthens commit time until quorum and ordering guarantees are satisfied.

A. Comparative Analysis of Cloud Replication Systems

Despite considerable progress, DynamoDB, FoundationDB, and GeoGauss continue to reveal structural limitations along the consistency–latency–availability frontier. DynamoDB's configurable consistency model achieves low latency but risks stale reads; strong reads are restricted to regional scope, and globally strong modes require wide-area coordination. Its otherwise lightweight two-of-three quorum inflates latency under synchronous multi-region deployment, while the three-replica design tolerates only a single failure, reducing write availability during WAN impairments [14]. FoundationDB delivers strict serializability yet relies on a multi-role control path involving commit proxies, resolvers, and log servers. This design increases coordination sensitivity and tail latency under load. Although failover is robust, sustaining cross-region safety typically demands five or more replicas, thereby elevating resource cost [16]. GeoGauss ensures strong global consistency through full replicas and epoch-based OCC ordering under Raft membership, but wide-area validation and full data movement in the commit path increase coordination overhead. While regional failures can be absorbed, partitions delay progress until global ordering and quorum are re-established, imposing the heaviest infrastructure burden due to per-region full replicas [17].

In our previous studies, BVAGQ found that write-query availability might be enhanced with minimal communication costs by employing a limited replication quorum [18]. Nonetheless, that methodology did not encompass cloud-native environments. This paper introduces the Binary Vote Assignment in Cloud (BVAC), a re-engineered variant designed to support synchronous replication across fragmented databases, with BVAGQ providing disjoint partitioning of data. BVAC enforces strong-by-default consistency by validating commits through vote-based quorums of constant size, thereby preserving serializability without incurring continuous wide-area coordination. Communication overhead is bounded by a fixed quorum of three to five servers, which reduces message complexity and constrains commit round-trip latency. Data availability is sustained through a binary vote assignment that admits multiple valid quorum paths, ensuring resilience under failures. In this way, BVAC retains the lightweight advantages of BVAGQ while extending its applicability to cloud infrastructures with explicit support for fragmentation and cross-region fault tolerance. The primary strategies for database fragmentation are horizontal fragmentation and vertical fragmentation. Horizontal fragmentation grants users' access to all attributes. Vertical fragmentation partitions the database based on properties, rather than entire rows [19, 20, 21] (Table I).

TABLE I. THIS SECTION ANALYSES EXISTING CLOUD DATABASE SYSTEMS

System Method	Data Consistency	Commit Path	Communication Cost
Dynamo DB	Configurable consistency model: default eventual consistency, optional strong consistency per request, with ACID serializable transactions.	- Primary replica handles write with 2-of-3 quorum - Strong reads from primary, eventual reads from any secondary. - Transactions require 2PC. - Async global tables reduce cost, while synchronous multi-region adds cross-region latency and coordination.	3 replicas per partition (regional); extra regions for global tables.
Foundation DB	Strict serializability through synchronous replication across primary and secondary roles.	- Strict serializability via synchronous replication. - Writes coordinated through commit proxies and multiple log servers (primary + secondaries), - Strong correctness and failover support, but higher coordination overhead and resource cost.	Often ≥5 replicas incl. satellites/coordinators (configurable).
GeoGauss	Strong global consistency through full-replica multi-primary with epoch-based OCC.	- Multi-primary architecture with full replicas in all regions. - Any primary can accept writes, but global commit order requires cross-region coordination. - Strong global consistency and resilience.	Full replica per region - high communication cost and cost overhead.

III. METHODOLOGY

The fundamental concept of replication is creating numerous copies of identical data or replicas across various storage locations. This research introduces the Binary Vote Assignment in Cloud (BVAC) method. In BVAC, all servers are systematically arranged in a two-dimensional grid configuration. If BVAC comprises twenty-five servers, they will be systematically arranged in a 5 x 5 grid format.

A. BVAC Algorithm Definition

In this section, the Binary Vote Assignment in Cloud (BVAC) is proposed by considering the distributed database fragmentation. The following notations are defined:

i. C is a table within the database.
ii. c' is the instance in C and C'
iii. $T(C)^1$ is the four servers in the corners
iv. $T(C)^2$ is the alternative locations on the peripheries
v. $T(C)^3$ is the central locations
vi. T is a transaction.
vii. x is a variable in C that gets altered by element of T.
viii. y is a variable in C that expected to remain unaltered by element of T.
ix. C_1 is a vertical fragmented table with data x.
x. C_2 is a horizontal fragmented table with data x.
xi. Pk is a primary key.
xii. Pk,x is a primary key with data x.
xiii. Pk,y is a primary with data y, where $y \neq x$

xiv. $C_{1_{(Pk,x)}}$ and $C_{1_{(Pk,y)}}$ are a horizontal fragmentation relation.

xv. η and ψ are groups for the transaction T.

xvi. $\lambda = \eta$ or ψ where it represents different transaction T (before and until get quorum).

xvii. T_η is a set of transactions that comes before T_ψ, while T_ψ is a set of transactions that comes after T_η.

xviii. D is a union of all data objects managed by all transactions T of BVAC.

xix. BVAC transaction element T_λ is an element either in different set of transactions T_η or T_ψ.

xx. wT_λ is write counter for the transaction.

xxi. $\hat{V}_{\lambda_x}$ is a transaction that is transformed from T_{λ_x}

xxii. T_{μ_x} represents the transaction feedback from a neighbour site.

xxiii. T_{μ_x} exists if either T_{λ_x} or $\hat{V}_{\lambda_x}$ exists.

xxiv. Successful transaction at primary site $T_{\lambda_x} = 0$ where $T_{\lambda_x} \in D$ (i.e., the transaction locked an instant x at primary). Meanwhile, successful transaction at neighbour site $T(\mu_x) = 0$, where $\mu_x \in D$ (i.e., the transaction locked a data x at neighbour).

xxv. $\left\lceil \frac{n}{2} \right\rceil$ is the greatest integer function (i.e., n=9, $\left\lceil \frac{9}{2} \right\rceil = 5$.

B. Data Replication in BVAC

The Algorithm 1 details the replication process for BVAC via BVAC Commit Coordination (BCC), showing how a commit initiated at a primary and a neighbour server is executed.

Algorithm 1: The BCC Algorithm: Data Replication

```
1  manage_bvaqgar_transaction ( )
2  {
3  while (InComplete)
4  do
5    while (transStat ≠ "Abort")
6    do
7          /* receive T_λx || V̂_λx || T_μx where λ = η, ψ either
       from client or any BTM of replica i ∈ T(C)*/
8      receive (client @ BCC of i ∈ T(C): T_λx || V̂_λx);
9      pid_T_λx = process id of T_λx;
10     log_tT_λx = login time of T_λx;
11          /*recognize replica task either to be as primary or
           neighbour processing for T_λx, λ = η, ψ */
12     switch {
13      case (receive(client @ BCC of neighbour: T_λx || V̂_λx||
       T_μx):
14       primary_replica_processing ( );
15                  break;
16              case (receive(BCC of primary: T_λx || V̂_λx)):
17                  neighbour_replica_processing ( );
18       break;
19       }
20     receive (BCC: V̂_λx, x$erv_Vote || V̂_λx, x$erv_Vote,
21      $_LCount,uT_λx);
22     if (x$erv_Vote = 1) then
23              Commit V̂_λx ;
24     endif
25     if (x$erv_Vote = 0);
26     Endif
27   /*On receiving transStat = "Abort" from other replica and
     needs to release its lock*/
28   if (receive (BCC: V̂_λx , ∈ T_η transStat, PrimaryID)) then
29     T_λx ∈ V_ψ =T_λx ; /*current T_λx becomeT_λx ∈ T_ψ */
30     V̂_λx =V̂_λx ∈ T_η ; /*V̂_λx ∈ T_η that BCC received will
     survive*/
31   Abort T_λx ;
32     Rollback;
33          T_λx = 1; /*Target Set is equal to 1, means primary
     already gets lock*/
34   endif
```

Every server comprises a primary data file. A server is either functional or nonfunctional, and the status (functional or nonfunctional) of each site is significantly uncorrelated with the others. When a site is functional, the data at the server is accessible; otherwise, it is inaccessible.

For example, in Fig. 1, data from site A (a) is duplicated to its adjacent servers B and D. Consequently, site A possesses three replica servers. Site E has four adjacent servers, which are B, D, F, and H. Therefore, site E possesses five replica servers. Simultaneously, data from site F is duplicated to servers C, E, and I, indicating that server F possesses four clones.

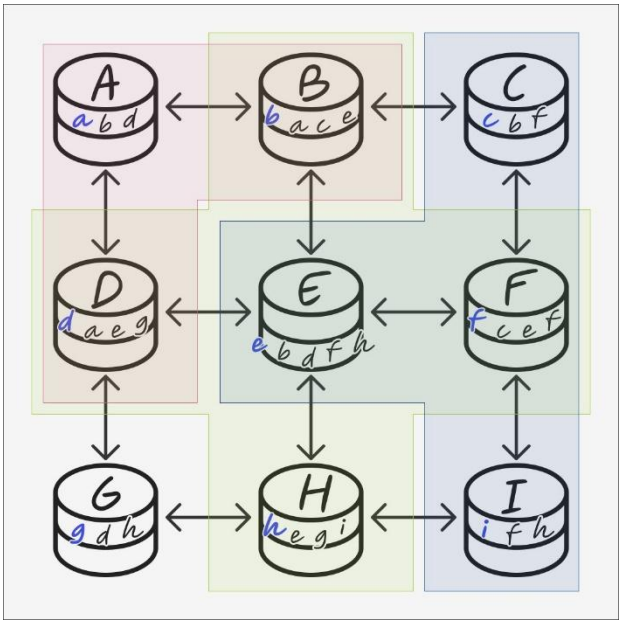

Fig. 1. BVAC framework.

The primary replica at a specific data x is the replica that acknowledges the client's request. In the BVAC algorithm, each replica of *T(C)* can simultaneously function as both a primary and a neighbor replica. Any replica $i \in T(C)$ may be designated as the primary replica, whereas additional replicas, $j \in T(C)$ where $i \neq j$ are considered neighbours. Line 7 processes a transaction T_{λ_x} that requests an update of data x from any replica of *T(C)*. Line 8-18 establish that the replica will serve as the primary, while the others will function as neighboring replicas to execute T_{λ_x}.

Recall our previous work, let *T(C)* be the set of replicas with replicated copies are stored corresponding to the assignment *C* for particular instant *x*, $T(C_x) = \{m(i,j), m(i-1,j), m(i,j-1), m(i,j+1), m(i+1,j)\}$. Two sets of transactions, T_η request instant *x* from *m(i,j)* replica, while T_ψ request instant *x* from *m(i-1,j)* respectively. The *m(i,j)* replica functions as the primary replica for processing T_η, where *m(i-1,j), m(i,j-1), m(i,j+1), m(i+1,j)* are neighbour replicas for processing $T_{\gamma_x} \in T_\eta$. Simultaneously, *m(i-1,j)* replica functions as the primary replica for processing T_ψ, while *m(i,j-1), m(i,j+1), m(i+1,j)* and *m(i,j)* are neighbour replicas for processing $V_{\gamma_x} \in T_\psi$. Both *m(i,j) and m(i-1,j)* replicas execute two different processing tasks concurrently. The *m(i,j)* replica is the primary replica managing T_η and its adjacent replica managing T_ψ, whereas the *m(i-1,j)* replica is the primary replica for processing T_ψ and neighbour replica for processing T_η. BVAC model considers different sets of transactions T_η and T_ψ. T_η is a set of transactions that comes before T_ψ, while T_ψ is a set of transactions that comes after T_η. The effect of BVAC transaction is defined as the processing of one instance of the transaction.

IV. Experimental Results

A. BVAC Algorithm Definition

To illustrate the operation of commit coordination under the BVAC Commit Coordination (BCC) mechanism, consider the case where two distinct transaction sets, T_η and T_ψ, concurrently request access to data file e at replicas E and B, respectively.

A cluster of five replication servers, interconnected as shown in Fig. 2, is used to illustrate the BVAC mechanism. Each primary replica propagates its database state to neighbouring replicas, allowing clients to access data from any server holding a replica. Consider two distinct transaction sets, T_η and T_ψ both requesting access to data file e at replicas E and B, respectively. When T_η and T_ψ attempt to update *e*, they must first issue update requests to their respective primary replicas, B and E. Both T_η and T_ψ propagate lock requests, but only the first transaction to acquire the lock proceeds, while the other is aborted. Consequently, replicas B and E maintain pending transactions, yet neither can read or update e concurrently. Primary node E, $T_{\eta,e} = 1$ propagates lock requests to neighbours B, D, F, and H, while primary node B $T_{\psi,e}$ propagates locks to neighbours E, D, F, and H. The transaction that first secures a majority quorum is transformed into $\hat{V}_{\lambda_x} \in V_\eta, V_\psi$. The details of experimental outcomes are summarized in Table II.

Assertion: If the transaction gets all locks from replica $i \in T(C)$, then the transactions will be executed successfully.

Proof: The only way that a transaction gets a lock in initiate lock is when $T_{\lambda_x} = 1$ with $T_{\lambda_x} \in T_\eta$. After $T_{\lambda_x} \in T_\eta$ success to initiate lock at a server, then, $T_{\lambda+1_x}$, ..., $T_{\lambda+q_x}$ which are the elements that exist in T_η will be queued. To get majority quorum, $wT_{\lambda_x} \geq \left\lceil \frac{n}{2} \right\rceil$ is required. This means that the primary server needs to get the majority locks from its neighbour servers by calling request lock from the neighbours servers. Each neigbours $i \in T(C)$ will send feedback to the primary to notify it is in free lock or not. If the primary gets the majority locks of instant *x*, it means that $\hat{V}_{\lambda_x} = T_{\mu_x} = T_\eta$ where $\hat{V}_{\lambda_x}$ gets a quorum. Next, the primary will send error notification to other neighbours $i \in T(C)$ in the quorum. Consequently, when $T_{\lambda_x} \in T_\psi$ releases its lock, $T_{\lambda_x} \in T_\eta$ gets the lock from every neighbour $i \in T(C_x)$. After T_{λ_x} gets majority quorum, relation *T* is fragmented into T_1 and T_2 using vertical fragmentation. Again, T_1 is fragmented into $T_{1_{(Pk,x)}}$ and $T_{1_{(Pk,y)}}$ using horizontal fragmentation. When a user finishes updating the instant, $\hat{V}_{\lambda_x}$ commit (send the fragmented data) to ∀ neighbour $i \in T(C_x)$. Therefore, all replicas of $T(C_x)$ will perform and execute the update successfully.

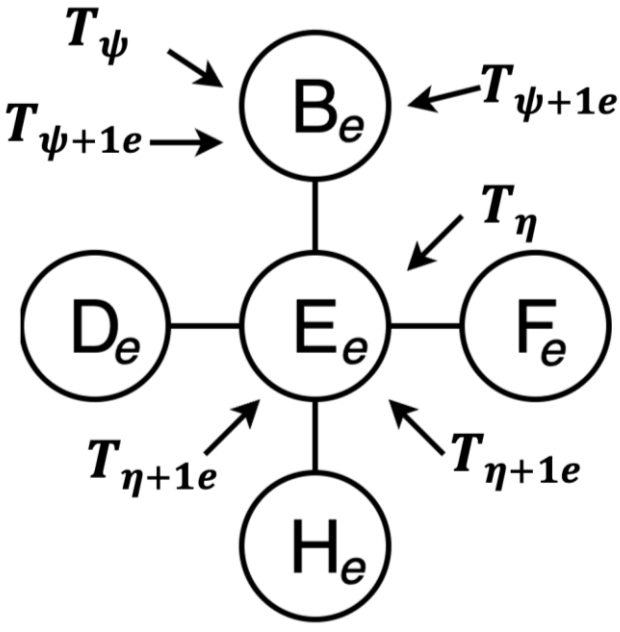

Fig. 2. An example of BVAC transaction requests.

TABLE II. AN EXAMPLE OF HOW BVAC HANDLE CONCURRENT TRANSACTIONS

REPLICA	E	B	D	F	H
TIME					
t1	Unlock (*e*)	unlock (*e*)	unlock (*e*)	unlock (*e*)	unlock (*e*)
t2	Request update				
t3	write l(*e*) counter_w(*e*)=1	Request update			
t4	wait	write lock(*e*) counter_w(*e*)=1			
t5	propagate lock:B				
t6	propagate lock:D	propagate lock:D			
t7			lock(*e*) from E		
t8	get lock:7 counter_w(*e*)=2	propagate lock:F			
t9	propagate lock:F			lock(*e*) from B	
t10	propagate lock:H	get lock:F counter_w(*e*)=2			
t11		propagate lock:H			lock(*e*) from E
t12	get lock:H counter_w(*e*)=3	propagate lock:E			
t13	obtain quorum release lock:3	propagate lock:H			
t14		abort $T_{\lambda_e} \in T_{\psi}$ & rollback, lock(*e*) from 8		rollback, lock(*e*) from E	
t15	update *e*				
t16	fragment *T' And T'*$_1$				
t17	commit $\hat{V}_{\lambda_e} \in V_{\eta}$	commit $\hat{V}_{\lambda_e} \in V_{\eta}$	commit $\hat{V}_{\lambda_e} \in V_{\eta}$	commit $\hat{V}_{\lambda_e} \in V_{\eta}$	commit $\hat{V}_{\lambda_e} \in V_{\eta}$
t18	Unlock (*e*)	Unlock (*e*)	Unlock (*e*)	Unlock (*e*)	Unlock (*e*)

B. Communication Cost Comparison

In this section, we compare the storage usage of BVAC, DynamoDB, FoundationDB, and GeoGauss. In BVAC, replication is bound to a fixed quorum size of three to five servers. This allows transactions to commit using a constant quorum, which limits the storage overhead even when the number of servers increases. In DynamoDB, each partition is synchronously replicated across three replicas within a region. When Global Tables are enabled, data is further replicated asynchronously across multiple regions, meaning the storage requirement increases proportionally with the number of regions. FoundationDB, by contrast, relies on a more complex configuration involving commit proxies, resolvers, and log servers. To ensure cross-region durability and strict serializability, the system often requires five or more replicas, including satellites, which elevates the storage footprint. GeoGauss employs the most storage-intensive approach, as it maintains a full replica of the database in every region. This design provides strong global consistency and resilience but comes at the cost of replicating the entire dataset across all regions. Table III shows the comparison between BVAC, DynamoDB, FoundationDB, and GeoGauss in terms of storage use.

From Table III, it is apparent that BVAC requires the least storage overhead by restricting replication to a small, constant quorum size of three to five servers. DynamoDB demands three replicas per partition in each region, and the storage requirement expands with the number of regions deployed in Global Tables. FoundationDB requires at least five replicas to preserve strict serializability and cross-region fault tolerance, thereby consuming more resources. GeoGauss incurs the heaviest storage cost since every region maintains a complete replica of the database. Consequently, BVAC demonstrates superior storage efficiency compared with DynamoDB, FoundationDB, and GeoGauss, making it a practical option for large-scale cloud systems where both consistency and resource optimization are critical.

TABLE III. STORAGE USAGE COMPARISON OF BVAC, DYNAMODB, FOUNDATIONDB, AND GEOGAUSS

Replication Techniques	Number of Servers	Storage Use
BVAC	3–5 replicas (constant quorum)	3–5
DynamoDB	3 replicas per partition (regional); extra regions for global tables	3 per region, grows with Global Tables
FoundationDB	≥5 replicas (primary, secondaries, satellites)	5 or more
GeoGauss	Full replica in every region	Equal to number of regions (high cost)

V. CONCLUSION

The rapid evolution of cloud computing has increased the significance of replication systems that can concurrently guarantee consistency, availability, and efficiency at scale. Conventional grid-based methodologies and commercial cloud platforms have advanced replication practices, but they continue to face inherent limitations within the consistency, latency and availability spectrum. This research presents the Binary Vote Assignment in Cloud (BVAC) as a cloud-native advancement of the BVAGQ replication model to tackle these ongoing challenges. BVAC integrates binary vote assignment into a quorum-based structure and utilizes the BVAC Commit Coordination (BCC) mechanism to authenticate transactions on fixed-size quorums across distributed replicas. This approach maintains robust consistency, minimizes communication complexity, ensures reliable fault tolerance with low latency, and improves storage efficiency by requiring fewer replicas than existing systems. Collectively, these contributions demonstrate BVAC as a scalable, reliable, and economical replication framework for extensive cloud settings. While the results of this study are promising, several aspects warrant further investigation. The current evaluation of BVAC is limited to specific workloads; its scalability and adaptability under highly dynamic and heterogeneous cloud conditions remain to be explored. In particular, BVAC could be extended with adaptive quorum resizing and workload-aware strategies to dynamically respond to shifting read/write ratios and latency requirements. Furthermore, future research should consider strengthening security and trust mechanisms, such as integrating Byzantine fault tolerance or lightweight blockchain-inspired auditing, to protect against adversarial conditions like malicious voting or quorum manipulation. Potential trade-offs, such as the complexity of quorum management, deployment across multi-cloud environments, and performance under real-world changeable workloads, need deeper analysis.

ACKNOWLEDGMENT

This research was funded by the Fundamental Research Grant Scheme – Early Career (FRGS-EC), Ministry of Higher Education Malaysia, under Grant Nos. FRGS-EC/1/2024/ICT08/UITM/02/3, 600-RMC/FRGS-EC 5/3 (053/2024). The authors also acknowledge the support of Universiti Malaysia Pahang Al-Sultan Abdullah under Grant No. RDU253002.

REFERENCES

[1] J. Eom, "Efficient data replication for fault tolerance in distributed computing environments," *Int. J. Distrib. Syst. Technol.*, vol. 8, no. 3, pp. 1–15, 2017.

[2] R. Dugyani and D. Govardhan, "Data replication and scheduling in the cloud with optimization assisted workflow management," Web Intelligence, vol. 22, no. 1, pp. 55–72, 2024.

[3] K. Acquah, "Empirical insights into replication models for distributed database environments," Int. J. Comput. Appl. (IJCA), vol. 186, no. 53, pp. 1–7, 2024.

[4] T. Taipalus, "Database management system performance comparisons: A systematic literature review," J. Syst. Softw., vol. 208, pp. 111–123, 2024.

[5] T. Hamrouni, R. Mokadem, and A. Khelifa, "Review on data replication strategies in single vs. interconnected cloud systems: Focus on data correlation-aware strategies," *Concurrency Computat.: Pract. Exper.*, vol. 35, no. 11, pp. e7612, 2023.

[6] A. Tahir, S. U. Khan, and N. Min-Allah, "Dynamic replication strategies in data grid systems: A survey," *Future Gener. Comput. Syst.*, vol. 111, pp. 654–667, 2020.

[7] S. Rehman, M. A. Jan, and S. Khan, "Survey on data replication in cloud systems," *Cluster Comput.*, vol. 25, pp. 1105–1124, 2022.

[8] M. M. Alshammari, A. A. Alwan, A. Nordin, and A. Z. Abualkishik, "Data backup and recovery with a minimum replica plan in a multi-cloud environment," *Procedia Comput. Sci.*, vol. 141, pp. 45–52, 2018.

[9] J. Alonso, R. Garcia-Castro, J. Cubo, et al., "Understanding the challenges and novel architectural models for multi-cloud native applications," *J. Cloud Comput.*, vol. 12, no. 1, pp. 1–22, 2023.

[10] G. Girau, "Towards adaptive replication policies for elastic cloud workloads," *IEEE Access*, vol. 12, pp. 45112–45126, 2024.

[11] A. Noraziah, A. A. C. Fauzi, S. H. S. A. Ubaidillah, B. Alkazemi, and J. B. Odili, "BVAGQ-AR for fragmented database replication management," *IEEE Access*, vol. 9, pp. 56168–56177, 2021, doi: 10.1109/ACCESS.2021.3065944.

[12] M. A. Fazlina, R. Latip, H. Ibrahim, and A. Abdullah, "Replication strategy with comprehensive data center selection method in cloud environments," *Computers, Materials & Continua*, vol. 74, no. 2, pp. 4139–4155, 2023

[13] A. Kaur, P. Gupta, M. Singh and A. Nayyar, "Data placement in era of cloud computing: A survey, taxonomy and open research issues," Scalable Computer, vol. 20, no. 2, pp. 377–398, 2019.

[14] V. Gupta, S. Kharche, A. Lakshman, S. Mittal, and R. Sumbaly, "Amazon DynamoDB: A scalable, predictably performant, and fully managed NoSQL database service," *Proc. USENIX Annu. Tech. Conf. (ATC)*, pp. 511–525, 2022.

[15] J. Idziorek, A. Keyes, C. Lazier, et al., "Distributed transactions at scale in Amazon DynamoDB," Proc. USENIX Annu. Tech. Conf. (USENIX ATC), pp. 675–690, 2023.

[16] Y. J. Zhou, M. Xu, A. Shraer, et al., "FoundationDB: A distributed unbundled transactional key-value store," Proc. ACM SIGMOD Int. Conf. Manag. Data (SIGMOD), pp. 2653–2666, 2021.

[17] G. Li, L. Zhou, Z. Zhang, et al., "GeoGauss: Strongly consistent and light-coordinated OLTP for geo-replicated SQL database," *Proc. ACM Manag. Data (PACMMOD)*, pp. 1–25, 2023.

[18] A. Noraziah, A. A. C. Fauzi, W. M. W. Mohd, et al., "Managing MyGRANTS fragmented database using Binary Vote Assignment Grid Quorum with Association Rule (BVAGQ-AR) replication model," Proc. Int. Conf. Data Eng. (DaEng-2015), Lect. Notes Electr. Eng., vol. 520, 2019.

[19] A. A. Che Fauzi, W. F. Wan Abdul Rahman, A. Fauzi, et al., "Managing fragmented database in distributed database environment," *J. Math. Comput. Sci. (JMCS)*, vol. 7, no. 1, pp. 8–14, 2021.

[20] M. Aggarwal, S. B. Bajaj, and V. Jaglan, "Performance analysis of degree of redundancy for replication in distributed database system," Proc. Int. Conf. Informatics (ICI), pp. 176–180, 2022.

[21] M. Goel and S. B. Bajaj, "Comparative analysis of vertical fragmentation techniques in distributed environment," *Int. J. Electr. Electron. Comput. Sci. Eng.*, vol. 5, no. 1, pp. 48–52, Feb. 2018.