GaussDB-Global: A Geographically Distributed Database System

Puya Memarzia*, Huaxin Zhang*, Kelvin Ho*, Ronen Grosman*, Jiang Wang†
*Huawei Technologies Ltd. Canada, †Huawei Technologies Ltd. China
Email: {*puya.memarzia1, *huaxin.zhang, *kelvin.ho, *ronen.grosman, †wangjiang16}@huawei.com

2024 IEEE 40th International Conference on Data Engineering (ICDE) | 979-8-3503-1715-2/24/$31.00 ©2024 IEEE | DOI: 10.1109/ICDE60146.2024.00383

***Abstract*—Geographically distributed database systems use remote replication to protect against regional failures. These systems are sensitive to severe latency penalties caused by centralized transaction management, remote access to sharded data, and log shipping over long distances. To tackle these issues, we present GaussDB-Global, a sharded geographically distributed database system with asynchronous replication, for OLTP applications. To tackle the transaction management bottleneck, we take a decentralized approach using synchronized clocks. Our system can seamlessly transition between centralized and decentralized transaction management, providing efficient fault tolerance and streamlining deployment. To alleviate the remote read and log shipping issues, we support reads on asynchronous replicas with strong consistency, tunable freshness guarantees, and dynamic load balancing. Our experimental results on a geographically distributed cluster show that our approach provides up to 14× higher read throughput, and 50% more TPC-C throughput compared to our baseline.**

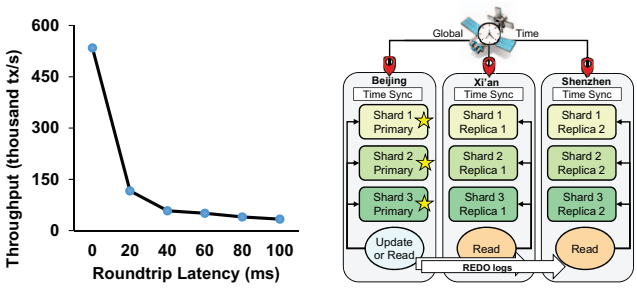
***Index Terms*—distributed database systems, replication, transaction management, query processing, high availability**

I. Introduction

Modern database systems are increasingly embracing geographically distributed architectures to support vast numbers of users across multiple regions. Geo-distributed systems rely on remote replication to support strong data availability and minimize data loss in the event of regional disasters. These systems are expected to provide services with high performance and minimal downtime.

Minimizing cross-region communication is the key to a fast geo-distributed system. Fig. 1a shows how Online Transaction Processing (OLTP) performance degrades as the system spans across more distant regions. Although decentralized mechanisms such as TrueTime [1] can alleviate timestamp assignment overhead, the cost of shipping redo logs (or deltas) across regions can be significant. Fig. 1b illustrates a real-world example with logs shipped between distant cities. In systems that use synchronous replication, write transactions wait on a quorum of replicas before they commit. If the quorum contains remote replicas, transactions must wait longer before they commit, causing performance degradation. Asynchronous replication avoids this issue by not waiting for data to reach distant replicas.

Read-only queries are typically routed to nearby replicas to reduce latency and improve performance. However, asynchronous replication poses new challenges because replicas may contain incomplete or inconsistent data. For sharded databases, each remote shard may have a different amount of



(a) TPC-C Performance Degradation (b) Global Time and Log Shipping

Fig. 1. Geo-distributed Database Overview

redo logs available thus reading the latest data on each replica can produce incorrect results. Customers may tolerate slightly stale data, but will never accept inconsistent data. There has been early research on this problem [2] but obtaining the freshest possible data while maintaining adequate performance and scalability is challenging. We tackle this issue with a global replica consistency point which guarantees correct reads on asynchronous replicas and can be computed quickly.

Communication overhead from a centralized transaction timestamp server can be avoided by using a decentralized, clock-based solution [1], [3]–[6]. Similarly, our system utilizes a high precision global clock mechanism to support transactions with external serializability. However, seamlessly migrating existing systems to decentralized transaction management is challenging due to fundamental differences in timestamp generation. Furthermore, strong dependence on time synchronization has implications for fault tolerance. Systems that use GPS clocks employ high redundancy [1]. In contrast, systems that use commodity hardware clocks pause the cluster if the clock-based mechanism fails [3], [6]. These issues motivated us to develop a novel zero-downtime bi-directional transition mechanism that streamlines deployment and maintenance, and strengthens fault tolerance.

In this paper, we present Huawei's GaussDB-Global (henceforth GlobalDB) as a novel solution for high performance at the geographic scale. We explain how GlobalDB achieves this using a decentralized transaction management system and describe its novel zero downtime transition between centralized and clock-based mechanism. We describe how GlobalDB improves performance further with reads on asynchronous local replicas using a global snapshot of the database. Lastly, we demonstrate with experiments that GlobalDB speeds up geo-

2375-026X/24/$31.00 ©2024 IEEE
DOI 10.1109/ICDE60146.2024.00383

distributed workloads with no impact to existing workloads.

Our key contributions are as follows:

- A decentralized transaction management system based on synchronized global clocks with a mechanism to seamlessly transition between centralized and decentralized transaction management.
- A flexible asynchronous replication scheme for sharded data that supports reads with guaranteed consistency, bounded freshness, and dynamic load balancing.

II. Background and Related Work

Our system is built on top of Huawei GaussDB [7]. In this section we provide some background knowledge on GaussDB, elaborate on the key mechanisms needed to build a geo-distributed system, and compare similar systems.

A. GaussDB Architecture

GaussDB is a distributed shared-nothing database system. A GaussDB cluster consists of computing nodes (CNs), data nodes (DNs), and a lightweight centralized transaction management system called Global Transaction Manager (GTM) which can scale out to over a thousand servers. The CN services client applications, parses queries, generates plans, and coordinates query execution on the DNs. DNs host portions of tables based on the distribution key's hash value or range. Replica DNs are typically placed at remote nodes for high availability. CNs are stateless and hence do not need replicas. The GTM provides timestamps for transaction invocation and commit. These timestamps are used for visibility checking through multi-version concurrency control (MVCC).

Primary data nodes continuously transmit updates to replica nodes in the form of Redo logs. A transaction may commit once its updates have been propagated to a quorum of replica nodes. If the quorum contains remote replicas, then the database can survive a site-level disaster. Alternatively, a transaction may commit earlier once its updates are replicated to some replicas in the same city. This provides some redundancy but does not protect against a regional disaster.

Although classic shared-nothing systems satisfy the business requirements for single region deployment, geo-distributed deployments pose new challenges. Data is moved or replicated to distant nodes to protect against regional disasters, provide better service to regional clients, and support businesses across geographic regions. Queries may involve expensive inter-node coordination, updates take longer to propagate to replicas, and nodes that are remote from the centralized transaction manager incur a much higher latency when fetching timestamps. We explore some of the related work on these issues.

B. Database Replication

Early research on replicated database reads dates back to the 1990s [2], [8], [9]. These approaches have difficulty scaling up due to synchronization overhead [2], [8], and transaction ordering and batching overhead [9]. Compared to those approaches, GlobalDB has a negligible performance impact on the primary data node, does not require a centralized log dispenser, and does not require fine-grained locking when applying Redo logs. Additionally, our system applies Redo logs in parallel which significantly improves log replay speed.

Replication schemes can be divided into two main categories based on how the logs are replicated: synchronous (also known as eager), and asynchronous (also known as lazy or optimistic). In systems with synchronous log replication, transactions wait until all or a quorum of nodes have persisted the update logs to disk [1], [3], [6], [10]. Synchronous log replication provides strong consistency at the cost of significantly higher update latency. Asynchronous log replication avoids waiting for replica nodes at the cost of weak/eventual data consistency and/or freshness, and a higher risk of data loss [11]–[13]. Some database systems can be configured to use either synchronous or asynchronous replication [7], [14], [15]. A third category of systems use epoch-based protocols [16], [17]. These systems group transactions within a small time window (the epoch) and defer synchronization with replicas until the epoch boundary. This reduces synchronization overhead compared to synchronous replication. However, aborts and long-running transactions penalize other transactions in the same epoch.

Compared to these systems, GlobalDB is the only sharded geo-distributed relational database with asynchronous physical replication, guaranteed consistency, adjustable freshness, and no negative impact on existing workloads.

C. Distributed Transaction Management

Database systems with centralized transaction management [7], [10], [15], [18], [19] are unsuitable for geo-distributed deployment due to high latency from cross-region communication and unavailability during regional failures. In light of this, modern systems use clock-based approaches to eliminate the overhead of a centralized transaction management system. Spanner [1] generates global timestamps using tightly synchronized satellite-connected GPS and atomic clocks with redundancies to provide high availability. GlobalDB employs the same approach for timestamp generation, and introduces a bi-directional transition protocol. In contrast, CockroachDB [3] and Yugabyte [6] synchronize clocks without the need for specialized hardware by using software services such as Network Time Protocol (NTP). These systems produce timestamps that are strictly monotonic using a Hybrid Logical Clock (HLC) [20] combining physical and logical (Lamport) time. Primary nodes append a Lamport timestamp to the commit Redo log indicating the maximum known timestamp of every other shard. Each replica checks if other replicas have applied up to the Lamport timestamp. This approach increases Redo log overhead but saves on deployment cost. Conversely, FaRMv2 [5] uses commodity hardware with local time that is synchronized at the data center level.

III. Global Clock in GaussDB

GaussDB's centralized transaction management can scale up to one thousand servers within a local cluster. However, high-volume timestamp traffic can still negatively impact other

types of data flow such as two-phase commits and primary-to-replica Redo log shipping, and remote network traffic degrades performance even further.

We tackle this issue by implementing a global-clock-based (henceforth GClock) algorithm for transaction management. The GClock algorithm is fundamentally the same as Spanner [1]. Both meet the following visibility requirements.

R.1 If trx_2 started after trx_1 committed with respect to global time, then trx_2 sees trx_1's updates.

R.2 If trx_1 has not committed before trx_2 started with respect to global time, trx_2 does not see trx_1's updates.

A transaction gets its GClock timestamp from its computing node's internal clock. The clocks need to be perfectly aligned with each other to satisfy visibility requirements. However, even synchronized clocks can drift apart over time. To resolve this issue, we deploy an accurate and reliable global time source device at each regional cluster. This device includes a GPS receiver and an atomic-clock and is capable of reporting time accurate to within nanoseconds of real time. Machines in the cluster synchronize their clocks with this local global time device every 1 millisecond. Clock deviation is low because synchronization is achieved within 60 microseconds as a TCP round trip, and the CPU's clock drift is bounded within 200 Parts Per Million (PPM) [5].

A GClock timestamp TS_{GClock} consists of clock time T_{clock} and an error bound T_{err} obtained from the clock synchronization network roundtrip T_{sync} and clock drift T_{drift}. The upper and lower bound time is thus obtained from $\text{T}_{clock} \pm \text{T}_{err}$.

$$\text{TS}_{GClock} = \text{T}_{clock} + \text{T}_{err} \qquad \text{T}_{err} = \text{T}_{sync} + \text{T}_{drift} \tag{1}$$

Transaction invocations and commits use the following protocol to obtain timestamps.

Invocation: Wait until $\text{T}_{clock} > \text{TS}_{GClock}$ and begin transaction. Single shard queries bypass this wait by using the node's last committed transaction timestamp.

Commit: Wait until $\text{T}_{clock} > \text{TS}_{GClock}$ and commit.

Following this timestamp protocol, GlobalDB meets the visibility requirements outlined in **R.1** and **R.2**, thus guaranteeing that all transactions are externally serializable.

GClock improves performance by reducing network and transaction management overhead. Additionally, GClock simplifies deployment at the geographic scale. Compared to prior clock-based transaction management systems, GClock provides a flexible transaction management system which supports both centralized and clock-based modes, and allows online transitioning between these modes without downtime.

A. Migration between GTM and GClock Modes

GTM and GClock transactions are inherently incompatible with each other due to different approaches to timestamp generation. GTM timestamps initially start from zero and increment by one per transaction.

$$\text{TS}_{GTM} = \text{TS}_{GTM} + 1 \tag{2}$$

In contrast, GClock timestamps use the current epoch time (currently a 10 digit number) which continuously increases even in the absence of new transactions. Even if we initialize GTM to the current time, it is possible for a new GTM transaction to have a smaller timestamp than an older GClock transaction due to the relatively slower growth of GTM timestamps. This presents a problem for migration because correctness requires all transaction timestamps to be monotonically increasing relative to their order.

The most straightforward way to tackle this issue is to block the system from accepting any new transactions and wait until all existing GTM-based transactions have finished and the global time has moved past the last timestamp assigned by the GTM Server. However, this solution entails significant system downtime which is unappealing to customers. Therefore, we propose an online migration approach that allows the co-existence of transactions using different timestamp generation methods. Our approach mitigates the anomalies that may arise from incompatible timestamps.

We address this issue in two scenarios: GClock-to-GTM transitions and GTM-to-GClock transitions. A GClock-to-GTM anomaly may occur when a new GTM transaction gets a smaller timestamp than a previously committed GClock transaction. Similarly, a GTM-to-GClock anomaly may occur when clock skew causes a new GClock timestamp to be smaller than a previously committed GTM timestamp.

We resolve these issues with a *DUAL* mode which acts as a bridge between GTM and GClock transactions and can co-exist with both. A DUAL mode timestamp TS_{DUAL} is guaranteed to be larger than both the most recent GTM timestamp TS_{GTM} and clock upper bound. During a transition, new transactions use DUAL mode as an intermediate step to avoid anomalies and satisfy visibility requirements. DUAL mode keeps the system online throughout the transition, continuously accepts new transactions, and maintains correctness.

$$\text{TS}_{DUAL} = \max(\text{TS}_{GTM}, \text{TS}_{GClock}) + 1 \tag{3}$$

The protocol to switch the cluster from GTM to GClock using DUAL mode is summarized in Fig 2. We begin by switching the GTM server (GTMS) and then each CN to DUAL mode. During this transition, the GTM server will service timestamp requests for both GTM and DUAL mode CNs. The GTM server also tracks the maximum issued timestamp and error bound until all nodes acknowledge the switch to DUAL mode. DUAL mode transactions first obtain a GClock timestamp and then communicate with the GTM server to receive a commit timestamp and a wait duration that avoids anomalies with existing GTM transactions. The GTM must remain in DUAL mode for $2\times$ the maximum error bound observed during the GTM to DUAL mode transition period. Only after this wait time can the cluster begin transitioning from DUAL mode to GClock mode, again starting from the GTM server followed by the CNs. All new transactions from this point on will use GClock mode. The wait time ensures that new GClock timestamps will be larger than all previous

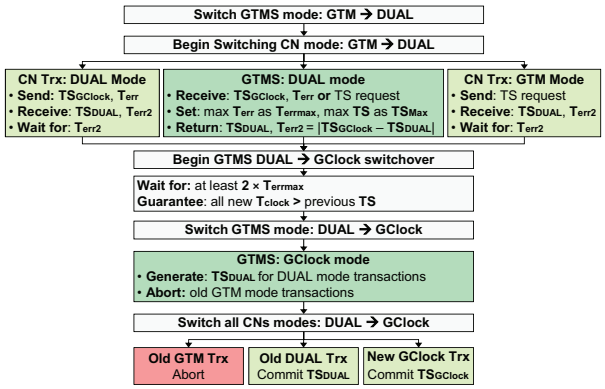


Fig. 2. GTM to GClock Transition using DUAL mode

timestamps. All running DUAL mode transactions can commit safely. Old GTM mode transactions that try to commit after the cluster has transitioned to GClock mode will abort.

During the transition, the GTM server is in DUAL mode but it is possible to have a combination of GTM, DUAL, and GClock transactions running concurrently because the CNs might not switch modes at precisely the same moment. A wait time is needed during DUAL mode to ensure correctness, otherwise the GTM-to-GClock transitions may cause visibility issues. Consider the following example.

Listing 1. Example showing why GTM transactions wait in DUAL mode

```
GTMS Running in DUAL mode
Node1 Running Trx1 in GTM mode
Node2 GTM mode → DUAL mode
Node3 GTM mode → DUAL mode
Node1 GTM mode → DUAL mode
Node2 DUAL mode → GClock mode
Node3 Send large GClock timestamp ts3 to GTMS
GTMS Raise internal timestamp to ts3
Node1 Trx1 gets large DUAL mode timestamp ts1 > ts3
      from GTMS and Commit Trx1 without waiting
Node2 Trx2 starts with timestamp ts2 < ts1
      Trx2 cannot see Trx1's committed update
```

To avoid this issue, GTM mode transactions must wait at commit if the GTM server is in DUAL mode. As shown in Fig 2, this wait time is double the largest error bound received by the GTM server during the transition.

Transitioning from GClock back to GTM mode is illustrated in Fig 3. This scenario may occur if there is a clock or synchronization issue. The system can safely switch to GTM mode and shift back to GClock mode once the issue is resolved. The logic for GClock-to-GTM transition is similar albeit slightly simpler than a GTM-to-GClock transition. The GTM server keeps track of the largest GClock timestamp issued so far. As a result, no old transactions will need to abort because the GTM server will issue timestamps that are larger than the largest GClock timestamp issued until the moment of transition. This also eliminates the need to wait while in DUAL mode, and nodes can begin switching to GTM mode as soon as all nodes have switched to DUAL mode.

IV. Reads on Replica

In GlobalDB, replica nodes can be used to fulfill read-only queries. This greatly reduces network latency if the client or

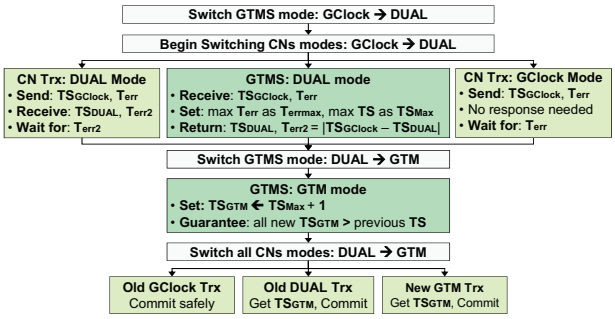


Fig. 3. GClock to GTM Transition using DUAL mode

the computing node is physically closer to the replica node than the primary node. If a primary node fails, its replica nodes can continue to serve read-only queries until the failed primary node recovers, or a replica node is promoted to replace the primary node. Allowing replica reads also helps to distribute the load from primary nodes to less busy replica nodes.

A. Read with Consistency at Replica

GlobalDB is a shared-nothing architecture where relations may be distributed across multiple shards. Each shard has a primary node that accepts read and write operations and one or more replica nodes that are read-only (Fig. 1b). Redo logs are shipped asynchronously, and the speed of applying Redo logs at each replica nodes may be different. As a result, it is possible for different shards of the same relation to have different levels of freshness at the replicas. Therefore, we need a consistent snapshot of the database, representing data that is available on all replicas at a point of time in history that is as close to the present as possible. We call this point in time the *Replica Consistency Point* (RCP).

Finding and maintaining the RCP is non-trivial and related research dates back to the early 1990s [8], [9]. Our approach involves finding the largest commit timestamp that is available on all replica nodes. Transactions committed before the RCP timestamp are visible, and any partial transactions and transactions with unfulfilled dependencies are invisible.

The example in Fig. 4 illustrates a scenario involving three different replicated shards. Each replica has a stream of incoming Redo logs with transaction commit timestamps ranging from ts_1 to ts_5 in chronological order. Using the algorithm, GlobalDB picks the maximum commit timestamps ts_4 from Replica 1, ts_5 from Replica 2, and ts_3 from Replica 3, shown circled in the figure. Thus the RCP timestamp is calculated as $min\{ts_4, ts_5, ts_3\} = ts_3$. All ROR queries will return transactions with a lesser or equal timestamp to ts_3, meaning Trx_1, Trx_2 and Trx_3 are the only visible transactions at this point. This is accurate because Trx_4 may have more than one shard involved, and we do not know if its Redo logs will arrive on Replica 2 or Replica 3. Trx_5 may depend on Trx_4 as it has a larger timestamp, so it cannot be visible. On Replica 1 Trx_1's commit timestamp is smaller than Trx_2's, meaning it does not depend on Trx_2, even if its Redo log appears after Trx_2's redo log. Therefore Trx_1, Trx_2 and Trx_3

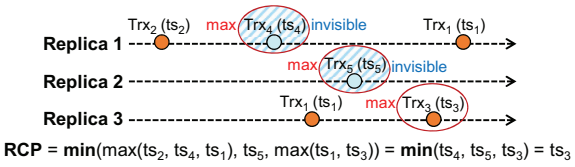


Fig. 4. Replica Consistency Point Calculation

are either single shard transactions, or committed, or pending their final commit Redo log as explained below.

Although commit timestamps increase monotonically, the order in which they are written to the Redo log is not necessarily ascending. This is because getting the timestamp from GClock or GTM and writing the timestamp as a commit Redo record may occur out of order due to thread context switching. Therefore, to provide correct visibility, we wait on tuples that are associated with in-progress transactions until they either commit or abort. This safe-guard is implemented by writing a special PENDING_COMMIT redo log at the primary before the transaction gets its invocation timestamp. This in turn locks the associated tuples on the replica node. Similarly, for two-phase commit transactions, a prepared transaction's visibility check at the replica is blocked until a COMMIT PREPARED or ABORT PREPARED record is replayed.

Each replica keeps track of its maximum commit timestamp. A CN is selected at the remote site to periodically collect the maximum timestamps from the replicas, calculate the RCP, and distribute it to other CNs. If the CN goes down, a different CN is selected to take over the RCP calculation. This approach has two benefits. First, it prevents the RCP from moving backwards from the perspective of client applications because clients may get routed to different CNs for reasons such as load balancing and failover. Second, it allows CNs to use remote replicas for reads. This is useful if the local replica is down, overloaded, or stale. Reading from a remote replica may still be faster than a remote primary. We describe this node selection algorithm in Section IV-B.

Not all transactions involve all data nodes, thus a replica node's maximum timestamp could lag behind when it does not receive any transactions to replay. A *heartbeat* transaction is periodically sent from the CN to all replicas to guarantee that the max commit timestamp always increases. From a client application's point of view, the RCP increases monotonically and consecutive ROR queries always show data with equal or greater freshness than previous queries.

When a Data Definition Language (DDL) statement (such as CREATE TABLE or DROP INDEX) is executed, it is expected to be visible and effective on subsequent queries. Due to the inherent delay in replaying logs to the replica nodes, we take additional measures to ensure the ROR queries are consistent with any relevant DDL statements. As such, we allow ROR queries if at least one of the following conditions are true:

1) The RCP is greater than the largest DDL timestamp. This means all DDLs have been replayed on all replicas. We skip the second check if this one passes.
2) The RCP is greater than the DDL timestamp for each table that is involved in the ROR query.

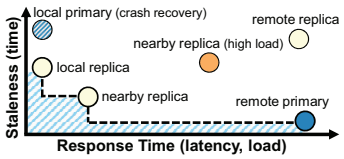


Fig. 5. ROR dynamic node selection using skyline

B. ROR Node Selection

In a geo-distributed cluster, the same data may be available from a multitude of nodes with varying levels of freshness, performance, and health. Reading from any replica located within the same region typically achieves the lowest network latency, but it does not distinguish between different replicas in the same region or consider each node's data freshness, load, and failure state. To solve this issue, we propose a dynamic node selection mechanism that picks the best nodes for each query based on per-node metrics that the CN tracks.

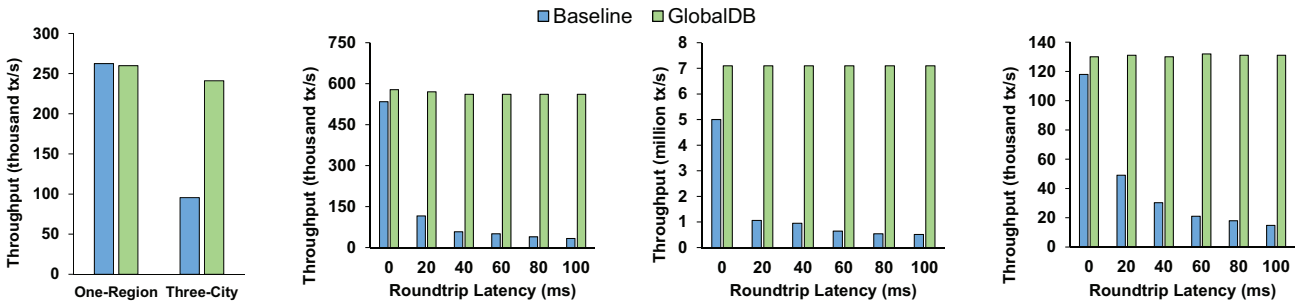
GlobalDB automatically detects failed or overloaded nodes and reroutes queries to other nodes. This rerouting is periodically done in the background, allowing GaussDB to achieve load balancing and respond to changes in node status. Fig. 5 illustrates our cost-based node selection using how much data a replica node has replayed (data staleness) and how promptly this node responds to queries (latency and load). Each computing node periodically refreshes this metric to form a skyline of candidate nodes. When running under GClock mode, replica staleness is measured by comparing the last committed transaction's timestamp against the current time. When running under GTM mode, we estimate the staleness based on the gap between the RCP and the last committed timestamp, and the rate at which new timestamps were issued during the last interval. Given a query with a bounded staleness requirement, the computing node picks a set of replicas from the skyline to answer the query with minimal latency.

The cost/staleness-based algorithm for picking replica nodes helps us to dynamically balance workload and provide high availability. For example, we may offload a busy primary node's reads to a replica node, or we may swap out a replica node for a different one if its response time goes up. When a node crashes, it is automatically excluded from the skyline.

V. Performance Evaluation

Our experimental setup consists of a cluster deployed within a single data center with simulated network delay (henceforth One-Region), and a geographically distributed cluster deployed in three different cities (henceforth Three-City).

The One-Region cluster consists of three Huawei ARM64 TaiShan servers all housed within the same server rack and connected via 10 gigabit Ethernet. Each server is equipped with two Huawei Kunpeng 920 [21] CPUs and 256GB of DRAM. The Three-City cluster includes a high precision time device in each region, and consists of three Intel x86 servers equipped with two Intel Xeon E5-2680 v2 CPUs, and 188GB of DRAM. The servers are located in Xi'an, Langzhong, and Dongguan, forming a triangle with 25ms, 35ms, and 55ms latency on each edge. Both clusters run EulerOS [22].



(a) TPC-C - Synchronous (b) TPC-C - Asynchronous (c) TPC-C - Modified (Read-only) (d) Sysbench Point Select

Fig. 6. Experimental results

Each cluster has a total of three computing nodes, six data nodes, and 12 replica nodes. We evaluate performance using TPC-C [23] and Sysbench [24]. Unless otherwise noted, we run the full TPC-C benchmark with all five transaction types, configured with 600 warehouses and 600 client terminals. The Sysbench experiments are configured with 250 tables with 25000 rows each and 600 client threads.

A. Transaction Management and Log Shipping

In this set of experiments we demonstrate GlobalDB's efficient transaction management and redo log shipping.

Real customer workloads have some degree of physical affinity. In light of this, we modify our workloads to control the proportion of remote transactions. To quantify improvement from each feature separately, we start with 100% local transactions to evaluate GClock. In this scenario, the sources of performance degradation are limited to transaction management and log replication. We later increase the percentage of remote transactions in Section V-B to evaluate ROR.

On the Three-City cluster, the network bandwidth between cities is considerably lower compared to the One-Region cluster. This increases transaction latency because Redo logs are buffered for longer before they can be transmitted. We examine this network overhead by switching GlobalDB to synchronous replication and running TPC-C. We also collocate the GTM server on the machine with the lowest mean latency to the other machines. The results in Fig. 6a show that the baseline's throughput decreases by about two thirds. GlobalDB eliminates transaction management overhead by using GClock, and improves network overhead by compressing the Redo logs with LZ4 compression [25], utilizing TCP BBR for more aggressive congestion control [26], and disabling Nagel's buffering algorithm to reduce receiver acknowledgement latency [27]. Once we deploy GlobalDB on the Three-City cluster, throughput increases to 91% of the One-Region cluster. Furthermore, GlobalDB does not suffer a performance penalty when directly deployed on the One-Region cluster.

In Figs. 6b, 6c, 6d we simulate some network delay on our One-Region cluster using Linux Traffic Control (tc) [28]. To emphasize the transaction management network overhead, we show the throughput of a node that is not co-located with the GTM server. In Fig. 6b we observe that baseline GaussDB's performance degrades by up to 90% when a 100ms network delay is added between the machines. GlobalDB achieves the same throughput regardless of network delay.

B. Read Performance

Read throughput emphasizes the benefits of GlobalDB's ROR feature. Here we modify TPC-C to only run the Order-status and Stock-level transactions thus turning it into a read-only benchmark. 50% of transactions are configured to be multi-shard. In Fig. 6c we see that TPC-C read throughput improves by up to $14\times$ on GlobalDB compared to the baseline due to reading from replicas and reduced transaction management overhead. In the Sysbench Point Select workload, 2/3 of the tuples are fetched from a remote node. As shown in Fig. 6d, GlobalDB improves Sysbench read throughput by up to $8.9\times$ over the baseline due to reading from local replicas.

VI. Conclusion

GaussDB-Global (GlobalDB) is a sharded, geographically distributed relational database system designed for high transaction throughput, low latency reads, and high availability. Our system achieves high performance using a decentralized clock-based transaction management system, reads from asynchronous replicas, and redo log shipping optimizations. GlobalDB guarantees external consistency using either decentralized or centralized transaction management. Our novel DUAL mode and bi-directional transition mechanism allows geo-distributed features to be activated on a live system without the need to take it offline. It also keeps the system fully operational in the event of a clock synchronization failure. GlobalDB's novel Read-On-Replica feature improves read performance with guaranteed consistency, adjustable freshness, and zero impact to write performance. Different replication schemes and different levels of disaster recovery provide flexibility to meet different customer requirements. Our experimental results show that GlobalDB can achieve performance that approaches a co-located cluster without any performance regressions for existing workloads. For future work, we are considering transparent load balancing based on geographical access patterns, self-assembling geo-distributed clusters to assist with deployment, and synchronous replicated tables that co-exist with asynchronous tables to meet specific business requirements by trading off update performance in favor of maximizing freshness and read performance.

REFERENCES

[1] J. C. Corbett, J. Dean, M. Epstein, A. Fikes, C. Frost, J. Furman, S. Ghemawat, A. Gubarev, C. Heiser, P. Hochschild, W. Hsieh, S. Kanthak, E. Kogan, H. Li, A. Lloyd, S. Melnik, D. Mwaura, D. Nagle, S. Quinlan, R. Rao, L. Rolig, Y. Saito, M. Szymaniak, C. Taylor, R. Wang, and D. Woodford, "Spanner: Google's Globally-Distributed database," in *10th USENIX Symposium on Operating Systems Design and Implementation (OSDI 12)*. Hollywood, CA: USENIX Association, Oct. 2012, pp. 261–264. [Online]. Available: https://www.usenix.org/conference/osdi12/technical-sessions/presentation/corbett

[2] C. A. Polyzois and H. Garcia-Molina, "Evaluation of remote backup algorithms for transaction processing systems," in *Proceedings of the 1992 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '92. New York, NY, USA: Association for Computing Machinery, 1992, p. 246255. [Online]. Available: https://doi.org/10.1145/130283.130321

[3] R. Taft, I. Sharif, A. Matei, N. VanBenschoten, J. Lewis, T. Grieger, K. Niemi, A. Woods, A. Birzin, R. Poss, P. Bardea, A. Ranade, B. Darnell, B. Gruneir, J. Jaffray, L. Zhang, and P. Mattis, "Cockroachdb: The resilient geo-distributed sql database," in *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 14931509. [Online]. Available: https://doi.org/10.1145/3318464.3386134

[4] Y. Li, G. Kumar, H. Hariharan, H. Wassel, P. Hochschild, D. Platt, S. Sabato, M. Yu, N. Dukkipati, P. Chandra, and A. Vahdat, "Sundial: Fault-tolerant clock synchronization for datacenters," in *Proceedings of the 14th USENIX Conference on Operating Systems Design and Implementation*, ser. OSDI'20. USA: USENIX Association, 2020.

[5] A. Shamis, M. Renzelmann, S. Novakovic, G. Chatzopoulos, A. Dragojević, D. Narayanan, and M. Castro, "Fast general distributed transactions with opacity," in *Proceedings of the 2019 International Conference on Management of Data*, ser. SIGMOD '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 433448. [Online]. Available: https://doi.org/10.1145/3299869.3300069

[6] (2020) Yugabytedb: Distributed sql database. Yugabyte, inc. [Online]. Available: https://www.yugabyte.com/

[7] (2023) Gaussdb distributed relational database system. Huawei. [Online]. Available: https://www.huaweicloud.com/intl/en-us/product/gaussdb.html

[8] R. P. King, N. Halim, H. Garcia-Molina, and C. A. Polyzois, "Management of a remote backup copy for disaster recovery," *ACM Transactions on Database Systems (TODS)*, vol. 16, no. 2, pp. 338–368, 1991.

[9] C. A. Polyzois and H. Garcia-Molina, "Evaluation of remote backup algorithms for transaction-processing systems," *ACM Transactions on Database Systems (TODS)*, vol. 19, no. 3, pp. 423–449, 1994.

[10] D. Huang, Q. Liu, Q. Cui, Z. Fang, X. Ma, F. Xu, L. Shen, L. Tang, Y. Zhou, M. Huang, W. Wei, C. Liu, J. Zhang, J. Li, X. Wu, L. Song, R. Sun, S. Yu, L. Zhao, N. Cameron, L. Pei, and X. Tang, "Tidb: A raft-based htap database," *Proc. VLDB Endow.*, vol. 13, no. 12, p. 30723084, aug 2020. [Online]. Available: https://doi.org/10.14778/3415478.3415535

[11] M. N. Vora, "Hadoop-hbase for large-scale data," in *Proceedings of 2011 International Conference on Computer Science and Network Technology*, vol. 1, 2011, pp. 601–605.

[12] M. Elhemali, N. Gallagher, N. Gordon, J. Idziorek, R. Krog, C. Lazier, E. Mo, A. Mritunjai, S. Perianayagam, T. Rath, S. Sivasubramanian, J. C. S. III, S. Sosothikul, D. Terry, and A. Vig, "Amazon DynamoDB: A scalable, predictably performant, and fully managed NoSQL database service," in *2022 USENIX Annual Technical Conference (USENIX ATC 22)*. Carlsbad, CA: USENIX Association, Jul. 2022, pp. 1037–1048. [Online]. Available: https://www.usenix.org/conference/atc22/presentation/elhemali

[13] P. Chairunnanda, K. Daudjee, and M. T. Özsu, "Confluxdb: Multi-master replication for partitioned snapshot isolation databases," *Proc. VLDB Endow.*, vol. 7, no. 11, p. 947958, jul 2014. [Online]. Available: https://doi.org/10.14778/2732967.2732970

[14] (2023) Mysql 8.0 reference manual. Oracle. [Online]. Available: https://dev.mysql.com/doc/refman/8.0/en/replication.html

[15] Z. Yang, C. Yang, F. Han, M. Zhuang, B. Yang, Z. Yang, X. Cheng, Y. Zhao, W. Shi, H. Xi, H. Yu, B. Liu, Y. Pan, B. Yin, J. Chen, and Q. Xu, "Oceanbase: A 707 million tpmc distributed relational database system," *Proc. VLDB Endow.*, vol. 15, no. 12, p. 33853397, aug 2022. [Online]. Available: https://doi.org/10.14778/3554821.3554830

[16] Y. Lu, X. Yu, L. Cao, and S. Madden, "Epoch-based commit and replication in distributed oltp databases," *Proc. VLDB Endow.*, vol. 14, no. 5, p. 743756, jan 2021.

[17] W. Zhou, Q. Peng, Z. Zhang, Y. Zhang, Y. Ren, S. Li, G. Fu, Y. Cui, Q. Li, C. Wu, S. Han, S. Wang, G. Li, and G. Yu, "Geogauss: Strongly consistent and light-coordinated oltp for geo-replicated sql database," *Proc. ACM Manag. Data*, vol. 1, no. 1, may 2023.

[18] (2023) Ibm db2. IBM. [Online]. Available: https://www.ibm.com/products/db2

[19] (2023) Mariadb. MariaDB Foundation. [Online]. Available: https://mariadb.org/

[20] S. S. Kulkarni, M. Demirbas, D. Madappa, B. Avva, and M. Leone, "Logical physical clocks," in *Principles of Distributed Systems: 18th International Conference, OPODIS 2014, Cortina dAmpezzo, Italy, December 16-19, 2014. Proceedings 18*. Springer, 2014, pp. 17–32.

[21] (2023) Taishan 2480 server. Huawei. [Online]. Available: https://e.huawei.com/en/products/computing/kunpeng/taishan/taishan-2480-v2

[22] M. Zhou, X. Hu, and W. Xiong, "openeuler: Advancing a hardware and software application ecosystem," *IEEE Software*, vol. 39, no. 2, pp. 101–105, 2022.

[23] (2010) Tpc benchmark c revision 5.11. TPC. [Online]. Available: https://www.tpc.org/TPC_Documents_Current_Versions/pdf/tpc-c_v5.11.0.pdf

[24] A. Kopytov. (2013) Sysbench homepage. [Online]. Available: https://github.com/akopytov/sysbench

[25] (2011) Lz4 - extremely fast compression. Collet, Y. [Online]. Available: https://github.com/lz4/lz4

[26] A. Toonk. (2020) Tcp bbr - exploring tcp congestion control. Medium. [Online]. Available: https://atoonk.medium.com/tcp-bbr-exploring-tcp-congestion-control-84c9c11dc3a9

[27] J. Nagle, "Congestion control in ip/tcp internetworks," *SIGCOMM Comput. Commun. Rev.*, vol. 14, no. 4, p. 1117, oct 1984.

[28] B. Hubert, T. Graf, G. Maxwell, R. van Mook, M. van Oosterhout, P. Schroeder, J. Spaans, and P. Larroy, "Linux advanced routing & traffic control," in *Proceedings of the Ottawa Linux Symposium*, sn. Ottawa, Ontario, Canada: Ottawa Linux Symposium, 2002, pp. 213–222.

[29] A.-C. Anadiotis, R. Appuswamy, A. Ailamaki, I. Bronshtein, H. Avni, D. Dominguez-Sal, S. Goikhman, and E. Levy, "A system design for elastically scaling transaction processing engines in virtualized servers," *Proc. VLDB Endow.*, vol. 13, no. 12, p. 30853098, aug 2020. [Online]. Available: https://doi.org/10.14778/3415478.3415536

[30] H. Avni, A. Aliev, O. Amor, A. Avitzur, I. Bronshtein, E. Ginot, S. Goikhman, E. Levy, I. Levy, F. Lu, L. Mishali, Y. Mo, N. Pachter, D. Sivov, V. Veeraraghavan, V. Vexler, L. Wang, and P. Wang, "Industrial-strength oltp using main memory and many cores," *Proc. VLDB Endow.*, vol. 13, no. 12, p. 30993111, aug 2020. [Online]. Available: https://doi.org/10.14778/3415478.3415537

[31] G. Li, X. Zhou, J. Sun, X. Yu, Y. Han, L. Jin, W. Li, T. Wang, and S. Li, "Opengauss: An autonomous database system," *Proc. VLDB Endow.*, vol. 14, no. 12, p. 30283042, jul 2021. [Online]. Available: https://doi.org/10.14778/3476311.3476380

[32] D. B. Lomet, "High speed on-line backup when using logical log operations," *ACM SIGMOD Record*, vol. 29, no. 2, pp. 34–45, 2000.

[33] K. Daudjee and K. Salem, "Lazy database replication with ordering guarantees," in *Proceedings of the 20th International Conference on Data Engineering*, ser. ICDE '04. USA: IEEE Computer Society, 2004, p. 424.

[34] (2023) Taishan 5280 server. Huawei. [Online]. Available: https://e.huawei.com/mx/products/servers/taishan-server/taishan-5280-v2

[35] (2023) opengauss relational database system. Huawei. [Online]. Available: https://gitee.com/opengauss/openGauss-server

[36] P. Chairunnanda, K. Daudjee, and M. T. Özsu, "Confluxdb: Multi-master replication for partitioned snapshot isolation databases," *Proc. VLDB Endow.*, vol. 7, no. 11, p. 947958, jul 2014. [Online]. Available: https://doi.org/10.14778/2732967.2732970

[37] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels, "Dynamo: Amazon's highly available key-value store," *SIGOPS Oper. Syst. Rev.*, vol. 41, no. 6, p. 205220, oct 2007. [Online]. Available: https://doi.org/10.1145/1323293.1294281

[38] X. Yan, L. Yang, H. Zhang, X. C. Lin, B. Wong, K. Salem, and T. Brecht, "Carousel: Low-latency transaction processing for

globally-distributed data," in *Proceedings of the 2018 International Conference on Management of Data*, ser. SIGMOD '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 231243. [Online]. Available: https://doi.org/10.1145/3183713.3196912

[39] I. Zhang, N. K. Sharma, A. Szekeres, A. Krishnamurthy, and D. R. K. Ports, "Building consistent transactions with inconsistent replication," *ACM Trans. Comput. Syst.*, vol. 35, no. 4, dec 2018. [Online]. Available: https://doi.org/10.1145/3269981

[40] K. Ren, D. Li, and D. J. Abadi, "Slog: Serializable, low-latency, geo-replicated transactions," *Proc. VLDB Endow.*, vol. 12, no. 11, p. 17471761, jul 2019. [Online]. Available: https://doi.org/10.14778/3342263.3342647

[41] Y. Lu, X. Yu, and S. Madden, "STAR," *Proceedings of the VLDB Endowment*, vol. 12, no. 11, pp. 1316–1329, jul 2019. [Online]. Available: https://doi.org/10.14778%2F3342263.3342270

[42] Y. Geng, S. Liu, Z. Yin, A. Naik, B. Prabhakar, M. Rosenblum, and A. Vahdat, "Exploiting a natural network effect for scalable, fine-grained clock synchronization," in *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*. Renton, WA: USENIX Association, Apr. 2018, pp. 81–94. [Online]. Available: https://www.usenix.org/conference/nsdi18/presentation/geng

[43] N. Cardwell, Y. Cheng, S. H. Yeganeh, and V. Jacobson, "Bbr congestion control," *IETF Draft draft-cardwell-iccrg-bbr-congestion-control-00*, 2017.

[44] G. Minshall, Y. Saito, J. C. Mogul, and B. Verghese, "Application performance pitfalls and tcp's nagle algorithm," *ACM SIGMETRICS Performance Evaluation Review*, vol. 27, no. 4, pp. 36–44, 2000.

[45] L. Lamport, "Paxos made simple," *ACM SIGACT News (Distributed Computing Column) 32, 4 (Whole Number 121, December 2001)*, pp. 51–58, 2001.