GeoLM: Performance-oriented Leader Management for Geo-Distributed Consensus Protocol

Duling Xu[†§], Dafang Zhang[†§], Tong Li[†*], Yunpeng Chai[†], Zegang Sun[†], Weiming Li[†], Yangfan Liu[†], Qipeng Wang[†], Jiaqi Liang[‡], Yang Ren[‡], Wei Lu[†], and Xiaoyong Du[†]
Renmin University of China[†], Huawei[‡],

***Abstract*—The global business of transnational enterprises demands geo-distributed databases, where the leader-follower-based consensus protocols are the key to guaranteeing consistency of replicas spread across regions. Compared with traditional databases running in a single data center, determining which node is the leader in consensus protocol has a greater performance impact in geo-distributed databases running across multiple data centers. However, the performance of legacy leader management is far from satisfactory due to the network and application dynamics (e.g., network delay, node popularity, operation read-write ratio). This paper proposes GeoLM toward performance-oriented leader management for geo-distributed consensus protocols. GeoLM captures the network and application dynamics and proactively conducts seamless leader handovers with bounded switching costs. Our geo-distributed experimental results show that GeoLM improves performance up to 49.75% over the baselines (e.g., Raft and Geo-Raft) and achieves considerably good performance compared to state-of-the-art consensus protocols (e.g., SwiftPaxos, CURP, and EPaxos).**

***Index Terms*—Geo-Distributed Database, Leader Management, Log Replication**

I. Introduction

Distributed databases, with their high scalability, high availability, and cost-effectiveness, are widely applied in fields such as cloud computing and big data analytics [1]–[21]. Traditional distributed databases are typically confined within a single data center [4]–[9]. However, with the increasing trend of data circulation in the digital economy era, transnational industries including banking, e-commerce, and cloud providers often operate multiple data centers. For instance, Huawei [10] and Amazon [11] span 93 and 105 availability data centers within 33 geographic regions, respectively. Recently, distributed databases have evolved from isolated services limited to single data centers towards collaborative services spanning multiple data centers. For example, Google BigQuery Omni [12] and Snowflake [13] connect data warehouses across Google Cloud, AWS, Azure, and other cloud providers, enabling real-time interactions during user queries across multiple clouds. Kubernetes Federation [14] connects multiple Kubernetes clusters over the Internet, where functions and data may reside in different regions or across different public cloud providers. In China, emerging companies like DatenLord [15], which implements distributed database systems ensuring strong consistency in data access across different data centers or cloud providers. These new business models are driving the development of a new paradigm for geo-distributed databases.

Consensus protocols such as Paxos [22], Raft [23], or their variants [24]–[34] have been widely used in modern distributed databases such as GaussDB [35], TiDB [36], PolarDB [37], Spanner [6], Aurora [38], and Neon [39]. These consensus protocols apply a leader-follower paradigm to guarantee replica consistency, where a leader node is responsible for handling client requests. In contrast, other nodes participate as followers to synchronize operation logs. The influence of leader selection on performance is negligible in traditional databases operating within a single data center. However, our study reveals that in geo-distributed databases, the role of the leader significantly impacts system performance (see §III-A).

Traditional consensus protocols perform well within the data center network [33], [40]. However, geo-distributed consensus protocol runs over a wide-area network (WAN) with various application requirements, where the best leader selection is challenging due to the *network dynamics* and *application dynamics*. First, the link delays between the leader to the followers significantly impact the log-replication latency. This reveals that leader selection should capture network dynamics in terms of network delays among all nodes. Second, selecting a leader node with more popular replicas or write operations reduces forwarding delays among nodes, improving system performance. This reveals that leader selection should capture application dynamics in terms of node popularity and read-write ratio of operations (see §III-B). However, the legacy ways of leader management typically focus either on network dynamics or application dynamics alone [25], [26]. In summary, the legacy leader management schemes are far from satisfactory for geo-distributed consensus protocols.

To tackle these issues, we introduce GeoLM[1], which considers both the network dynamics and application dynamics and proactively conducts seamless leader handovers with bounded switching costs. **First**, GeoLM periodically selects a target node according to a utility function of link delays, node popularity, and operation read-write ratio. This ensures GeoLM's network and application awareness. **Second**, GeoLM adjusts the leader switching window where the target node's utility remains positive. This avoids leader-switching oscillations and

This work is supported by the NSFC Project (No.62202473,62441230, No.U23A20299, and No.6241101193), the CCF-Huawei Populus Grove Fund (No.CCF-HuaweiDB202308). [§]The first two authors contributed equally to this work. [*]Tong Li is the corresponding author (tong.li@ruc.edu.cn).

[1]GeoLM is named after Geo-distributed Leader Management.

bounds GeoLM's overall switching costs. **Third**, GeoLM allows the current leader node to pre-synchronize logs with a target node that has fallen behind. This capability can aid the target node in winning the election and ensuring consistency in the consensus protocol.

We have implemented the GeoLM prototype in a distributed database based on ETCD and integrated it into the maintained Raft protocol [41]. GeoLM extends the leader management module without affecting the execution processes of the storage layer and database layer, thus ensuring compatibility with log-replication-based variants. We then conduct evaluations through both trace-driven simulation and testbed deployment. Evaluation results show that GeoLM reduces average/worst-case transaction latency over the baselines Raft by up to 30.77%/49.75% and Geo-Raft [27] by up to 23.97%/24.62%, respectively. GeoLM also achieves considerably good performance compared to the most recent *leader-management-based* approach SwiftPaxos [26] and the state-of-the-art *log-replication-based* approaches such as CURP [28] and EPaxos [31]. This serves as a strong validation of GeoLM.

The paper is organized as follows. §II overviews the related work. §III introduces the background and motivates GeoLM. The overview of GeoLM is depicted in §IV. The design and implementation are elaborated in §V. In §VI, we evaluate GeoLM. Finally, §VII concludes the paper.

II. Related Work

This section provides an overview of methods to enhance the performance of geo-distributed consensus protocols.

Optimization on leader management. As for leader management, several variants based on Paxos/Raft are proposed to enhance the communication efficiency of the consensus protocol [24]–[26], [42]. For example, Xu et al. [24] propose Raft-Plus to use inter-node network delay for leader selection. Vukolic et al. [25] proposed the Droopy/Dripple to strategically configure the leader ensemble based on historical workload and system response latency to better fit uneven workload conditions. Ryabinin et al. [26] propose SwiftPaxos, which changes the election process using two-round voting, accelerating the election process in case of replica divergence. These works focus either on network dynamics (e.g., Raft-Plus) or application dynamics (e.g., Droopy/Dripple, SwiftPaxos) alone.

Optimization on log replication. In terms of log replication, the fundamental optimization strategy involves enhancing performance through reducing communication requirements [27]–[34]. Among them, Geo-raft [27], CURP [28], and Pigpaxos [29] fall into the category of introducing new roles (e.g., secretary, observer, relay) to ease leader performance bottlenecks. Fast Paxos [30], EPaxos [31], Dpaxos [32], CockRoachDB [33], and GeoGauss [34] fall into the category of reducing the long-distance communication times to shorten global database transaction time. These studies, however, demand intrusive modifications to the logic of log replication, limiting their applicability. For example, CURP and EPaxos rely on the assumption of commutativity of operations and adapt to limited scenarios. Essentially, these variants operate independently of our proposed GeoLM, which can seamlessly integrate with these variants to enhance performance further.

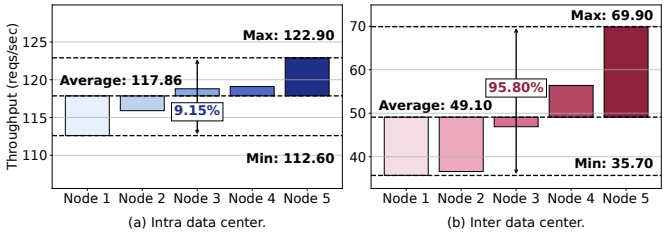


Fig. 1: An example of performance change when changing the leader node over intra-datacenter and inter-datacenter links. The X-axis represents the leader node.

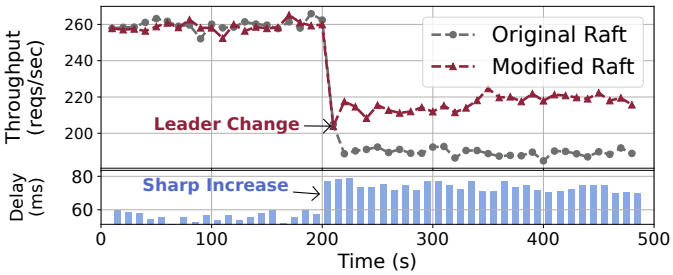


Fig. 2: An example of how delays impact performance.

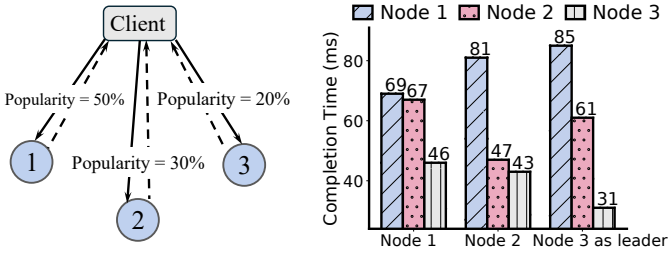
III. Background and Motivation

A. Leader Selection is More Crucial in Geo-Distributed Consensus Protocols

In a consensus protocol with the leader-follower paradigm, the leader handles client communication and coordinates the replication and synchronization of data replicas on other follower nodes. In general, choosing a better node as the leader positively correlates with performance. However, compared with traditional consensus protocol running in a single data center, leader election in geo-distributed consensus protocol has a greater performance impact. To explain this more clearly, we modified ETCD-Raft [41] to give an example of performance change when changing the leader node over intra-datacenter and inter-datacenter links. Figure 1 shows the results. The impact of leader selection on performance is unremarkable for traditional databases running in a single data center (a better leader only improves 9.15% performance). However, when nodes are spread across regions in a geo-distributed database, the system performance is more crucial to who is the leader node (a better leader improves 95.80% performance). This reveals that leader selection is more crucial in geo-distributed consensus protocols.

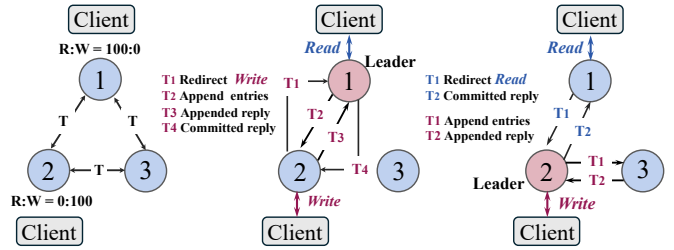
B. Leader Selection Should Capture Network Dynamics and Application Dynamics

When operating a geo-distributed consensus protocol, we see multiple challenges in determining the best leader selection. We will elaborate next.



(a) Popularity (b) Completion time under different node as leader.

Fig. 3: An example of how node popularity impacts performance. (a) shows an example of a cluster with different popularity. (b) shows the time required for each node in the cluster to complete its requests when the cluster leader is controlled (from left to right, Node 1, Node 2, and Node 3 as the leader).



(a) Example of nodes (b) Leader only reads (c) Leader only writes

Fig. 4: An example of how the read-write ratio impacts performance. The read-write ratio on Node 1 is 100:0, and on Node 2 is 0:100. The red text markers represent the time required for write requests, and the blue text markers represent the time for read requests.

Observation #1 (Network Dynamics): Leader selection depends on network delays among all nodes. Geo-distributed consensus protocol runs over a WAN whose dynamics are non-negligible [43]. The *network dynamics* is manifested in three aspects. (a) Enlarged value of delays. The round-trip time (RTT) of an inter-data-center link (e.g., 100 ms) is over three orders of magnitude higher than that of an intra-data-center link (e.g., 10 us). (b) Enlarged difference of delays. The RTT differences among nodes are tens to hundreds of milliseconds (e.g., RTT = 20 ms vs. RTT = 200 ms), which is non-negligible compared with the node processing delay. (c) Enlarged fluctuation of delays. The optimal leader may become sub-optimal due to RTT fluctuation caused by network congestion or route change. Figure 2 gives an example of performance change when changing the leader node under network deterioration (a sharp increase in the total delay among all nodes). We modified the ETCD-Raft to change its leader node at 200 s. Figure 2 shows that the performance impact from delay deterioration can be significantly reduced. This reveals that leader selection should consider network delays among all nodes. Geo-distributed consensus protocol also serves applications with dynamic characteristics. The *application dynamics* is manifested in **two aspects:** node popularity and read-write ratio of workload. We discuss it next.

Observation #2 (Application Dynamics): Leader selection depends on node popularity. Node popularity refers to the measure of how frequently a node is accessed or utilized for operations such as reads or writes. Under identical network conditions, nodes assuming the role of leader experience performance variations based on their respective levels of node popularity. Specifically, selecting a leader node with more popular replicas reduces forwarding delays among nodes, improving the system performance. To explain this more clearly, we show an example of three nodes with the same network delays (i.e., RTT = 20 ms) but with different node popularity (i.e., 50%, 30%, and 20%, respectively) in Figure 3. It has been observed that when Node 1 (with 50% popularity) acts as the leader, the latency is 34.7% lower compared to when Node 3 (with 20% popularity) acts as the leader. This reveals that leader selection should prioritize nodes with a higher popularity.

Observation #3 (Application Dynamics): Leader Selection depends on the read-write ratio of operations. The access pattern can fluctuate over time due to shifts in application usage, business needs, or seasonal demands. Where read operations can be handled directly by either the leader or any followers while write operations require a leader to coordinate proposals among followers, achieving agreement through a quorum, and ensuring durability through replicated logs. Under identical network conditions, nodes assuming the role of leader experience performance variations based on their read-write ratio of operations. Specifically, selecting a leader with more write operations reduces forwarding delay from followers to the leader, improving system performance. To explain this clearly, Figure 4 (a) shows an example of 3 nodes with the same network delays (i.e., RTT = 2T) but with different read-write ratios (i.e., the read-write ratio on Node 1 is 100:0, and the read-write ratio on Node 2 is 0:100). When Node 1 acts as the leader (see Figure 4 (b)), the latency is doubled compared to when Node 2 acts as the leader (see Figure 4 (c)). This reveals that leader selection should prioritize nodes with a lower read-write ratio.

C. Challenges

How to determine the target node? A target node is the node that has the potential to be the best leader. Our observations have shown that leader selection should consider network dynamics (i.e., network delay) and application dynamics (i.e., node popularity, read-write ratio). Generally, the system performance gain after a node becomes the leader is positively related to a utility model, where the utility function should simultaneously consider factors including network delay, node popularity, and read-write ratio. In this paper, GeoLM determines the target node if it achieves the highest utility value.

How can the target node be voted the leader? The target node with the highest utility may lose the election during the election process. This can be attributed to the outdated log of the target node. To ensure the target node receives votes from a majority of the nodes, the log of the target node should be

up-to-date before sending out request-to-vote RPCs (Remote Procedure Calls). In this paper, GeoLM proactively switches the leader regardless of whether the current leader fails or not, distinguishing it from legacy consensus protocols that only initiate leader changes in response to leader failures. To ensure the target node is voted as the leader, GeoLM takes full advantage of the present leader node to pre-synchronize logs where the target node has fallen behind.

How to control the frequency of leader switching? A small frequency of leader switching may fail to respond promptly to network and application dynamics. In contrast, frequent leader switching may induce excessive switching costs, degrading the system's performance. Assuming that each leader switching requires a fixed cost, the total cost induced by GeoLM is proportional to the switching frequency. In this paper, GeoLM adopts a damping mechanism to dynamically adjust the leader switching window to control the frequency of leader switching. The goal is to adapt to network and application dynamics with bounded switching costs.

IV. GeoLM Overview

The primary goal of GeoLM is to achieve consistently high performance by switching the leader node according to the network and application dynamics. In this section, we first model the utility function for each node following its assumption of leadership, and then we introduce the framework of GeoLM based on the utility model.

A. Utility Modeling

We define the node utility u as the performance improvement when it becomes the leader. For a distributed database consisting of N nodes, the utility of a node n is:

$$u_n = \sum_{i=1}^{N} P_i \cdot (C_i^{leader} - C_i^n) \tag{1}$$

where P_i refers to the popularity of node i, C_i^{leader} represents the completion time of requests reaching node i with the present leadership, and C_i^n represents the completion time of requests reaching node i when node n acts as the leader.

According to the way of handling read and write operations in consensus protocols, C_i^{leader} and C_i^n can be further computed as follows:

$$C_i^{leader} = R_i \cdot T_i^{leader} + W_i \cdot (\tau_{leader} + T_i^{leader}) \tag{2}$$

where R_i and W_i refer to the read ratio and write ratio of operations, respectively. Particularly, we have $R_i + W_i = 1$. T_i^{leader} represents the RTT between node i and the leader node, τ_{leader} represents the commit latency of node i with the present leadership. Similarly, we have

$$C_i^n = R_i \cdot T_i^n + W_i \cdot (\tau_n + T_i^n) \tag{3}$$

where T_i^n represents the RTT between node i and n, τ_n represents the commit latency of node i with the leadership of node n.

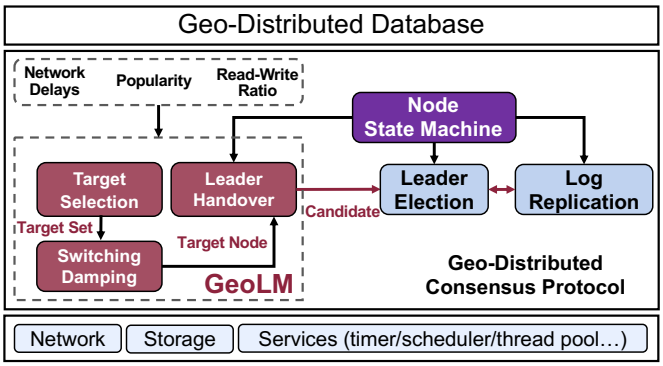


Fig. 5: The framework of GeoLM.

When substituting Equations (2) and (3) in (1), the utility of node n can be modeled as follows:

$$u_n = \sum_{i=1}^{N} P_i * [(T_i^{leader} - T_i^n) + W_i \cdot (T_{median}^{leader} - T_{median}^n)] \tag{4}$$

where T_{median}^{leader} denotes the median link delay from the leader to all followers, and T_{median}^n denotes the median link delay from node n to all other nodes. Here we argue that that applying the median link delay T_{median} to approximate the commit latency τ is reasonable. This is because the leader must wait for confirmation from at least half of all followers.

B. GeoLM Framework

Figure 5 illustrates the general framework of a geo-distributed consensus protocol, where GeoLM adds three components (colored with red blocks) namely: (1) Target Selection, (2) Switching Damping, and (3) Leader Handover. These additional components cooperate with the existing modules (colored with purple and blue blocks) in the legacy consensus protocol, aiming to achieve high-performance leader management for geo-distributed databases.

Target Selection. The input of this component is the RTT matrix between each node, the popularity of each node, and the read-write ratio of operations. These states are continuously updated at the leader node via interaction with follower nodes. According to Equation (4), the component first calculates the utility values for all nodes. Next, it identifies nodes with utility values exceeding 0 to compose a designated collection (termed the target set) as the output. This ensures GeoLM's network and application awareness (see §V-A).

Switching Damping. The target set may contain multiple optional nodes. The utility values of nodes may also vary dynamically (e.g., sometimes being greater than 0 and other times less than 0). This may result in *leader-switching oscillations*. To tackle this issue, GeoLM introduces a component, called Switching Damping, to select a stable node with a positive and high utility value. Specifically, the component dynamically adjusts the leader switching window (i.e., *observation period*) where the target node's utility value should be kept to exceed 0. This also controls the frequency of leader switching and bounds GeoLM's overall switching costs (see §V-B).

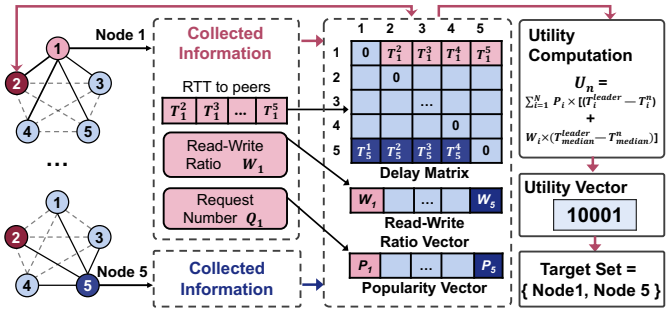


Fig. 6: An example of target selection workflow.

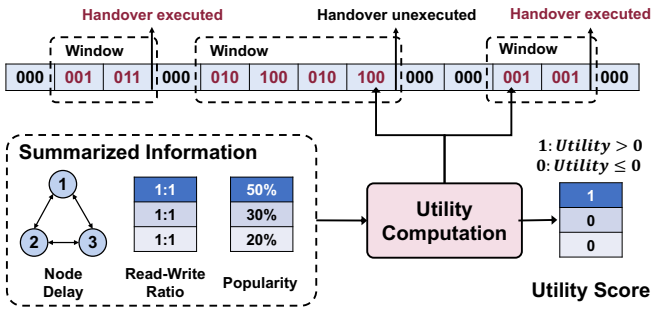


Fig. 7: An example of switching damping.

Leader Handover. Before the target node becomes the candidate for leader election, the Leader Handover component enables the present leader node to pre-synchronize logs where the target node has fallen behind. Specifically, GeoLM applies a modified *Node State Machine* where the present leader acts as the *Helper Node* for proactive leader switching. This can help the target node win the election and ensure the consistency of the consensus protocol (see §V-C).

V. System Design Details

This section elaborates on the design and implementation of the specific modules in GeoLM.

A. Target Selection

Figure 6 gives an example of a five-node geo-distributed database to better explain the workflow of the Target Selection.

First, for network delays, we've integrated application-layer RTT estimation into the consensus protocol to adapt to network dynamics. First of all, each node measures the smoothed RTT from this node to all other nodes via the heartbeat messages. Then the leader collects all the RTT vectors from each follower via the *RttProbeResp* messages. For instance, the RTT vector of Node 1 is $< T_1^2, T_1^3, T_1^4, T_1^5 >$. These RTT vectors ultimately converge at the leader node, where it is reshaped into a delay matrix as shown in Figure 6.

Second, the read-write ratio should be updated according to longer-term statistics where each follower counts the number of read-operations (e.g., query) and write-operations (e.g., update and insert) on each node and periodically syncs the read-write ratio to the leader. For instance, the leader can maintain a write-ratio vector $< W_1, W_i, ..., W_5 >$, where W_i denotes the write ratio of Node i.

Third, the node popularity cannot be estimated independently by a single node. Instead, it requires the followers (e.g., Node i) to periodically send the number of local requests (from clients, denoted by Q_i) to the leader. The leader then calculates each node's popularity accordingly. For instance, the popularity vector is $< P_1, P_i, ..., P_5 >$, where P_i denotes the popularity of Node i.

Finally, according to Equation (4), we calculate the node utility as the performance improvement when it becomes the leader. All nodes with positive utility enter the target set for selection, such as Nodes 1 and 5 in Figure 6.

B. Switching Damping

The design rationale of switch damping is to increase the difficulty of switching if the handover is executed successfully and reduce the difficulty of switching if the handover is unexecuted for a period. We define this period as the *damping window* (denoted by wnd), during which we decide whether to enter the leader handover. Assume the initial window is wnd_0 (typically proportional to the heartbeat cycle, see Figure 12(b) for more details), during the damping window, we check if there is at least one node whose utility value remains consistently positive (i.e., $u > 0$). If yes, we double the damping window (i.e., $wnd = 2 * wnd$) and enter the leader handover phase (as we will discuss in the next section); Otherwise, we halve the damping window (i.e., $wnd = min\{\frac{wnd}{2}, wnd_0\}$) and keep the leader node unchanged. Switch damping can control the switching frequency, thereby achieving a bounded switching cost (see §VI-C for more details).

To explain this more clearly, Figure 7 further gives an example of how the Switch Damping component works. Assume the initial window is $wnd_0 = 2$. Then we collect 2 utility vectors $< 0, 0, 1 >$ and $< 0, 1, 1 >$ and find that the utility of Node 3 keeps positive during the damping window. Then the damping window is doubled (i.e., $wnd = 4$) and Node 3 begins acting as the role of the *candidate* node. After the leader is switched to Node 3, we have to monitor 4 utility vectors $< 0, 1, 0 >$, $< 1, 0, 0 >$, $< 0, 1, 0 >$, and $< 1, 0, 0 >$ instead to check if there is any node whose utility value keeps positive during the damping window. In the example of Figure 7, we find that none of the nodes meet the conditions (i.e., the target set is $\emptyset$). In this case, the damping window will be halved (i.e., $wnd = 2$).

C. Leader Handover

The Leader Handover component modifies the leader election process by introducing a scheme called Log Pre-Synchronization. Specifically, we allow the leader to proactively retrieve the latest log index from the target node and update the logs accordingly, ensuring that the target node has the same maximum log index as the leader at the current term. This approach ensures that the target can more easily get votes after initiating an election. After the leader sends the log message to the target node, it steps down to become a helper and stops sending heartbeats. When the target node gets the log completion message, it already knows the leader

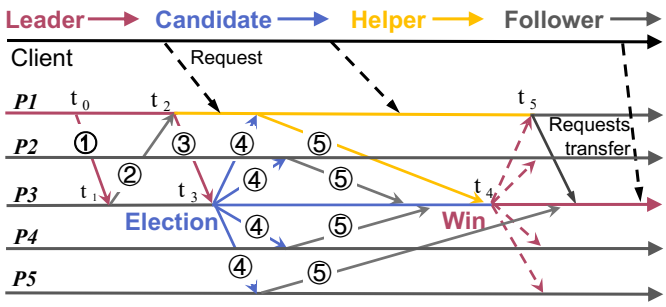


Fig. 8: An example of leader handover.

has stepped down and initiates an election without waiting for the *heartbeat timeout* (usually 150-300 ms [41]). This significantly reduces the cluster's downtime caused by waiting for the timeout, ensuring the target has a higher priority in initiating the election.

To explain this more clearly, we give an example of the entire handover process in Figure 8. **Step** ①: At time t_0, the Leader Node (*P1*) decides to transfer leadership to *P3* and sends *Msg.1* (as Message 1, the same below) to *P3* to request its latest log index. **Step** ②: P3 receives *Msg.1* at t_1 and responds to *P1* with *Msg.2*, which carries *P3*'s latest log index. **Step** ③: At t_2, after receiving the response, *P1* sends *Msg.3* to *P3*, containing all the committed logs at the leader (from *P3*'s latest log index to *P1*'s latest log index at t_2), then steps down to become a Helper Node, stopping heartbeats and beginning caching client requests. *P3* receives the committed logs from *P1* at t_3 and begins to catch up. **Step** ④: *P3* then becomes the Candidate Node and starts an election by sending *Msg.4* to request votes. **Step** ⑤: At t_4, the other Follower Nodes respond to the vote requests with *Msg.5*, casting their votes. Eventually, *P3* obtains the majority of votes and successfully becomes the new leader.

It is worth noting that another follower (other than the target node) might also reach the heartbeat timeout and initiate an election, winning the election before the target node enters the leader handover phase. In this case, GeoLM falls back to the legacy way of leader election. Upon receiving a heartbeat from the new leader, the Helper Node transitions to a Follower Node and forwards all client requests from its term, ensuring seamless handover without downtime.

D. Implementation

We implemented GeoLM in a distributed database based on ETCD [41] and integrated it into the maintained Raft protocol with less than 500 lines of code. Specifically, we reuse the already existing messages such as *Msg.Heartbeat*, *Msg.Vote*, and *Msg.Granted*. *Msg.Heartbeat* is used to report the heartbeat, *Msg.Vote* (i.e., *Msg.4* in Figure 8) is sent from the candidate to request votes, and *Msg.Granted* (i.e., *Msg.5* in Figure 8) is sent by the follower/helper to grant votes. We further define several new types of messages such as *Msg.RttProbe*, *Msg.RttProbeResp*, *Msg.LeaderHandover*, *Msg.LeaderHandoverResp*, and *Msg.LogsCompletion*. The *Msg.RttProbe* messages are

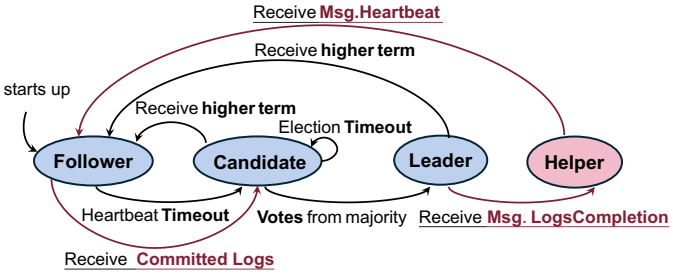


Fig. 9: The node state machine in GeoLM.

State	Vote	Receive requests	Cache requests	Send Msg.Heartbeat	Log Pre-Synchronization	Process requests
Leader	×	√	√	√	√	√
Follower	√	×	×	×	×	×
Helper	√	√	√	×	√	×

Fig. 10: A table of node state functions.

broadcasted periodically from each node in an all-to-all paradigm, and the *Msg.RttProbeResp* messages are sent to return timestamps for RTT sampling. Messages *Msg.LeaderHandover* and *Msg.LeaderHandoverResp* (i.e., *Msg.1* and *Msg.2* in Figure 8) are sent when the leader decides to transfer leadership. *Msg.LogsCompletion* message is sent by the candidate to transition the leader into a helper.

The GeoLM implementation also modifies the Node State Machine of the consensus protocol. As illustrated in Figure 9, we introduce helper, a new node state in which the prior leader node helps to cache requests before a new leader wins in the election process. Particularly, when receiving *Msg.LogsCompletion* message from the candidate, the leader transitions to the helper. When receiving *Msg.Heartbeat* message from a new leader, the helper transitions to the follower. During log pre-synchronization, when receiving the pre-synced committed logs from the leader, the follower (i.e., target node) transitions to the candidate. Figure 10 further lists the functions of nodes under different states. Particularly, the helper stops sending *Msg.Heartbeat* and begins caching requests instead of processing requests.

VI. Evaluation

A. Experiment Setup

Setup. We evaluate GeoLM by prototyping it on our trace-driven simulation and real-world testbed deployment.

• **Trace-driven simulation:** We utilize local servers with Traffic Control (TC) netem [44] for our testing. To accurately simulate complex network scenarios, including irregular absolute delay variations and jitter, we base our simulations on the trace data of the worldwide Amazon AWS cloud from an open website [45]. By applying the tool of Piecewise Cubic Hermite Interpolating Polynomial (PCHIP) [46], we generated over 10,000 multi-dimensional link delay matrices (10x10) that reflect the average link delays and distribution patterns among the geo-distributed AWS servers. These delay matrices are fed into TC to simulate the time-varying delays between data nodes in our geo-distributed database.

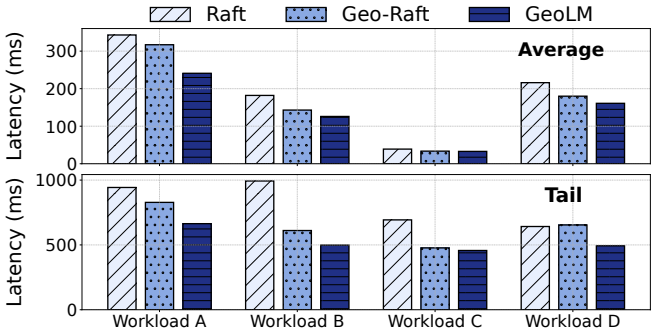
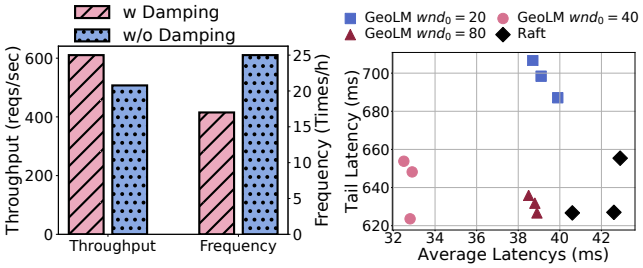


Fig. 11: Overall performance.



(a) Damping vs. w/o damping. (b) Parameter sensitivity.

Fig. 12: (a) Effectiveness of switching damping. (b) Performance under different initial damping windows (wnd_0) with the unit of heartbeat cycle in GeoLM (1 heartbeat cycle = 200 ms), and Raft (default).

• **Real-world testbed deployment:** We also deploy a seven-node geo-distributed database across various regions to assess performance in a real production environment. The server locations and configurations are as follows: one server in Guangzhou, two servers in Beijing (located in two different data centers), and two servers in Shanghai (also in two different data centers). Each server uses the HUAWEI Cloud c3.xlarge.2 instance with Ubuntu 22.04 Server 64-bit, and is configured with 4 CPU cores, 8GB of RAM.

Workloads. We conducted tests using the Yahoo! Cloud Serving Benchmark (YCSB) [47] and focused on workloads A, B, C, and D as primary test loads. Workload A consists of 50% reads and 50% updates. Workload B is made up of 95% reads and 5% updates. Workload C consists entirely of reads (100%) with a uniform access distribution. Lastly, Workload D comprises 95% reads and 5% inserts, maintaining a read-write ratio similar to Workload B while evaluating data insertion.

Schemes. We conduct a comparative performance evaluation of GeoLM with state-of-the-art consensus protocols. First, since GeoLM is integrated into Raft [23], it and Geo-Raft [27] are considered as baseline. We also compare GeoLM with the state-of-the-art Raft/Paxos variants including SwiftPaxos [26], CURP [28], and Epaxos [31]. Among them, SwiftPaxos is a representative leader-management-based approach to enhance performance. CURP and Epaxos are two representative log-replication-based approaches to enhance performance, whereas Epaxos further represents the leaderless consensus protocol.

B. Overall Performance

Method. Overall performance is measured using the YCSB benchmarking tool, where 20 clients simultaneously send requests to the system, completing 100,000 operations in 120,000 records. We monitor the average latency per request. Rigorously and fairly, we compared GeoLM with two baseline schemes, Raft and Geo-Raft, which were both uniformly implemented in the same ETCD project and compared under the same network traces and workloads.

Results. The subfigure at the top portion of Figure 11 indicates that GeoLM reduces the average transaction latency over Raft by 29.74%, 30.77%, 15.64%, and 25.46% for Workloads A, B, C, and D, respectively. In comparison to Geo-Raft, GeoLM demonstrated enhancements of 23.97%, 11.89%, 2.08%, and 10.56% for Workloads A, B, C, and D, respectively. We also examined performance under worst-case conditions, specifically the (99^{th} percentile) tail latency. As illustrated in the bottom portion of Figure 11, GeoLM's tail latency improved over Raft by 29.59%, 49.75%, 34.05%, and 23.21% for Workloads A, B, C, and D, respectively. When compared to Geo-Raft, the tail latency improvements are 19.81%, 18.33%, 4.19%, and 24.62% for Workloads A, B, C, and D, respectively. This demonstrates that the performance of geo-distributed transaction processing can be greatly improved by GeoLM. Additionally, we conducted experiments with more data nodes (e.g., 9 nodes), and the results remained consistent.

Analysis. First, the improvement in average and tail latency is most significant for Workload A when comparing both Raft and Geo-Raft. This enhancement is due to GeoLM, which considers the read-write ratio of nodes and allocates network resources more effectively for write operations that require more round trips, thereby boosting overall performance. **Second**, performance improvements are relatively smaller under all-read workload (i.e., Workload C). This limitation arises because the optimizations for write operations in GeoLM cannot be utilized in these scenarios. Instead, only basic performance enhancements can be achieved by selecting more efficient leader nodes.

C. GeoLM Deep Dive

This section first investigates the optimality, effectiveness, and cost of the key components (i.e., Target Selection, Switching Damping, and Leader Handover) in GeoLM. Then we discuss the performance impact of factors such as the read-write ratio and popularity. Finally, we compare GeoLM with more state-of-the-art Raft/Paxos variants.

Optimality of Target Selection. We assess the optimality of the Target Selection component by introducing two metrics: *precision* and *recall*. A precision rate of 100% indicates that all selected target nodes are correct, meaning that these nodes were optimal at the time of their selection based on subsequent performance results. Recall measures the extent to which we comprehensively select the correct optimal nodes and serves as an auxiliary metric. We tested GeoLM in a fixed seven-node cluster setup with consistent read-write ratios, trace-based latency, and node popularity characteristics while monitoring

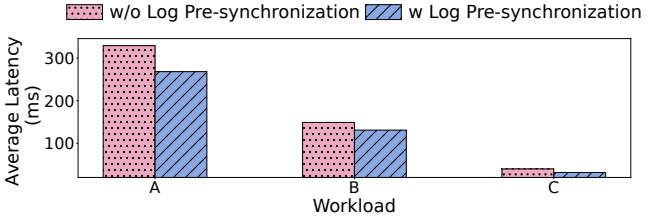


Fig. 13: Effectiveness of Log Pre-Synchronization.

each node's actual status and overall system performance. We compared the performance data of each node when it served as the leader without switching roles. Over three experiments, each lasting about one hour, we observed 20 instances of leader handovers. Of these, all 20 were correct, while 6 potentially correct nodes were not chosen, resulting in a precision rate of 100% and a recall rate of 76.92%. This indicates that our algorithm is highly accurate in node selection, and our conservative approach considers the error risks during handover, ensuring beneficial decision-making.

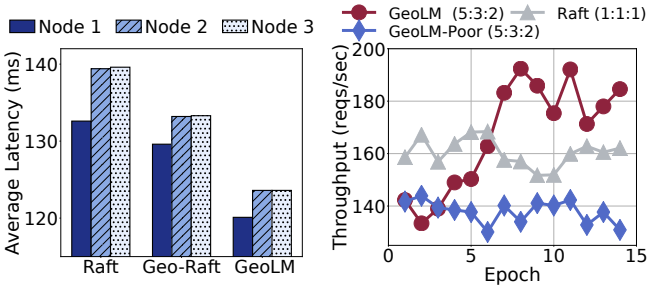
Effectiveness of Switching Damping. In this experiment, we assessed the effectiveness of damping by controlling whether the damping module was active and observing the performance under both conditions. As shown in Figure 12(a), we validated that damping is indeed effective in improving throughput(up to 20.34% better than not using) in GeoLM. Additionally, we observed an interesting phenomenon: although we theoretically bounded the switching frequency (relative to the number of windows) in §V-B, the initial window size, wnd_0, significantly impacts performance in practice. Specifically, if the window size is too large, it results in too few switches within the total completion time, leading to suboptimal performance. For example, in Figure 12(b), when wnd_0 is 80 times the heartbeat cycle, its average latency is worse than when wnd_0 is 40 times the heartbeat cycle. Conversely, if the window size is too small, such as wnd_0 being 20 times the heartbeat cycle, it results in an excessively high switching frequency, incurring additional costs. This manifests as no significant improvement in average latency but an increase in tail latency. Based on our implementation tests, we recommend using an initial window size of 40 heartbeat cycles. As shown by the pink square in Figure 12(b), this configuration achieves a balanced performance in both average and tail latency.

TABLE I: Switching duration analysis.

	Waiting Time	Election Duration	Total Duration
Raft	150 - 300 ms	234 ms	384 - 534 ms
GeoLM	30 us	153 ms	154 ms

Cost of Leader Handover. Table I gives a breakdown of the switching duration of GeoLM. We analyze hundreds of leader switching and find that the target node in GeoLM initiates an election within 30 us after receiving the *Msg.logCompletion*. The average switching duration is 153 ms. In contrast, Raft needs to wait for a heartbeat timeout of 150-300 ms to begin an election, with an average switching duration of 234 ms. Our total switch time saves at least 59.90% per switching.

Effectiveness of Log Pre-Synchronization. We independently tested the impact of log pre-synchronization during handover on overall performance. As shown in Figure 13, using pre-synchronization significantly reduces average response time under all types of workloads. This advantage becomes particularly pronounced as the write ratio increases (from Workload C to A). This is because a higher write ratio requires more log synchronization, thus amplifying the benefits of log pre-synchronization. Consequently, log pre-synchronization enhances overall performance and achieves greater performance gains under a higher write ratio.



(a) Read-write ratio (b) Popularity

Fig. 14: (a) Performance impact of read-write ratio. The read-write ratio for Node 1 is 50:50, Node 2 is 95:5, and Node 3 is 100:0. (b) Performance impact of popularity. GeoLM-Poor represents a GeoLM variant without considering the node's popularity. GeoLM (5:3:2) represents the full implementation of GeoLM under the condition that the read/write requests to Nodes 1, 2, and 3 are in proportions of 50%, 30%, and 20%, respectively.

Performance impact of read-write ratio. We simulate different practical application scenarios by controlling the read-write ratio of the nodes while keeping the cluster latency and popularity conditions consistent. Specifically, we set the read-write ratios of Node 1, Node 2, and Node 3 to 50:50 (read-write balance), 95:5 (more reads and less writes), and 100% read (i.e., read-only), respectively. We test the performance of each node in each cluster (Raft, Geo-Raft, and GeoLM) under such extreme read-to-write ratio differences, specifically showing the average latency in Figure 14(a). We observed varying overall performances among the three solutions under identical cluster read-write ratio settings, particularly in handling uneven read-write ratios. Among them, Raft exhibits the weakest performance in managing such disparities. Geo-Raft shows some improvements overall, attributed to the backup of secretary nodes, though these gains are not substantial. In contrast, GeoLM demonstrates intelligent handling of uneven load distribution, seamlessly transitioning the leader node and achieving significant performance enhancements. Specifically, GeoLM performs approximately 10.76% better than Raft and about 7.27% better than Geo-Raft in these scenarios.

Performance impact of popularity. In this experiment, we adjust the user request number for 3 nodes while keeping other variables consistent to validate the effectiveness of GeoLM regarding popularity. GeoLM-Poor represents a GeoLM variant without considering the node's popularity. GeoLM (5:3:2) represents the full implementation of GeoLM under the condition that the read/write requests to Nodes 1, 2, and

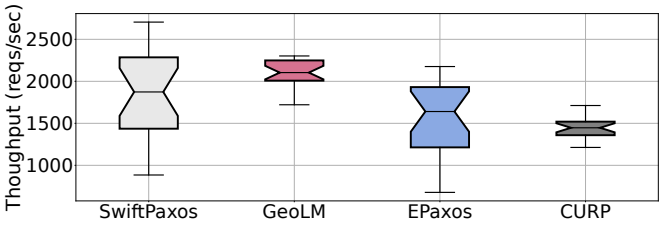


Fig. 15: Performance comparison with more schemes.

3 are in proportions of 50%, 30%, and 20%. GeoLM-Poor (5:3:2) represents the GeoLM-Poor under the condition that the read/write requests to Nodes 1, 2, and 3 are in proportions of 50%, 30%, and 20%, respectively. Raft (1:1:1) represents the Raft under the condition that the read/write requests are uniformly distributed. Figure 14 (b) shows the results. We find that throughput in Raft (1:1:1) distributed is stable. In the context of non-uniform requests, the system throughput is initially lower than Raft cluster and consistently remains low. This is due to the redirection of requests before processing. For GeoLM (5:3:2), the initial throughput is low, but as the epochs progress (1 epoch = 10 min), the throughput gradually increases, approaching and even surpassing the performance of Raft (1:1:1). The reason for this improvement is that, despite the differing request popularity levels for each node and the potential need for many request redirections, the popularity optimization eventually influenced the switching of the leader node. From the logs, we observe that by the second epoch, the system already completed the leader switching. Subsequently, the system throughput of GeoLM (5:3:2) significantly improves, surpassing the performance of both GeoLM-Poor (5:3:2) and Raft (1:1:1).

Comparison with more schemes. We further conduct experiments to compare GeoLM with a classic leaderless distributed consensus protocol EPaxos, the most recent single-leader consensus protocol SwiftPaxos, and a primary-backup replication protocol CURP based on the public repository [32]. We try our best to ensure these schemes run under similar conditions, including network fluctuations, popularity, and read-write ratios. Figure 15 indicates that GeoLM achieves considerably good performance compared to EPaxos, SwiftPaxos, and CURP, thus effectively validating the capabilities of GeoLM. It is worth noting that although EPaxos approaches its best performance, GeoLM's worst-case performance is far superior. This reveals that GeoLM exhibits better stability.

D. Real-World Testbed Deployment

We deploy GeoLM in two representative scenarios to investigate the GeoLM's performance over WAN links.

Two-region-three-center deployment: We deploy three cloud servers across two regions: two servers in different data centers in Beijing and one in Guangzhou, forming a two-region-three-center cluster. This type of cluster represents the most commonly deployed scheme in primary-backup solutions, such as Huawei's primary-backup database systems, known for its resilience against the failure of a data center [48]. As illustrated in Figure 16, the final results demonstrate that GeoLM outperformed both Raft and Geo-Raft significantly. Specifically,

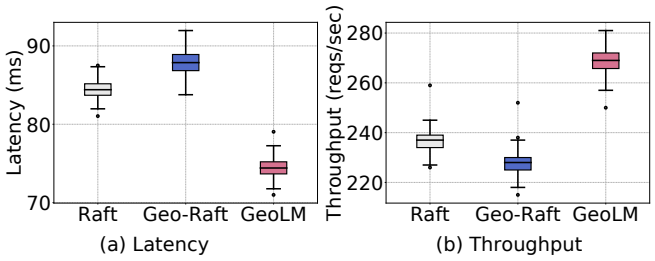


Fig. 16: Two-region-three-center scenario.

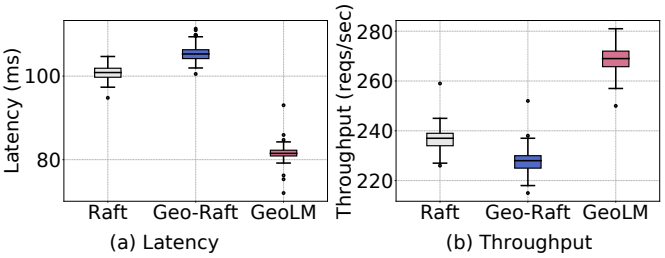


Fig. 17: Three-region-five-center scenario.

GeoLM achieved a 13.43% and 17.93% improvement in throughput compared to Raft and Geo-Raft, respectively. In terms of average latency, GeoLM exhibited 15.26% lower latency than GeoRaft and 11.89% lower latency than Raft. For tail latency, GeoLM showed reductions of 21.12% and 18.41% compared to GeoRaft and Raft, respectively.

Three-region-five-center deployment: We deploy five cloud servers across three regions: one in Guangzhou, two in different data centers in Beijing, and two in different data centers in Shanghai, forming a three-region-five-center cluster, Which represents the most widely discussed and researched scheme in academia, to withstand the failure of an entire region. As depicted in Figure 17, the results are as follows: In terms of throughput, GeoLM outperforms GeoRaft by 28.95% and Raft by 23.27%. In terms of average latency, GeoLM shows 22.59% lower latency than GeoRaft and 19.02% lower than Raft. In terms of tail latency, GeoLM demonstrates reductions of 26.45% compared to GeoRaft and 26.52% to Raft.

VII. Conclusion

Leader management becomes significantly more crucial in geo-distributed databases than in databases confined to a single data center. We propose GeoLM, which, for the first time, integrates both network and application dynamics to proactively optimize leader switching. This significantly enhances transaction performance while ensuring consistency and bounding switching costs. In the future, we plan to open-source GeoLM to drive innovation in the field of geo-distributed consensus protocols.

Acknowledgments

We thank the reviewers for their valuable suggestions. We thank Feng Zhang, Ju Fan, Yueguo Chen, Mengxing Liu, and Zheng Chen for their guidance and support. We are also grateful for conversations with and feedback from Zhanhuai Li and Guoliang Li.

REFERENCES

[1] D. Borthakur, J. Gray, J. S. Sarma, K. Muthukkaruppan, N. Spiegelberg, H. Kuang, K. Ranganathan, D. Molkov, A. Menon, S. Rash *et al.*, "Apache hadoop goes realtime at facebook," in *Proceedings of the 2011 ACM SIGMOD*, 2011, pp. 1071–1080.

[2] X. Zhang, H. Wu, Z. Chang, S. Jin, J. Tan, F. Li, T. Zhang, and B. Cui, "Restune: Resource oriented tuning boosted by meta-learning for cloud databases," in *Proceedings of the 2021 international conference on management of data*, 2021, pp. 2102–2114.

[3] Z. Chen, F. Zhang, J. Guan, J. Zhai, X. Shen, H. Zhang, W. Shu, and X. Du, "Compressgraph: Efficient parallel graph analytics with rule-based compression," *Proceedings of the ACM on Management of Data*, vol. 1, no. 1, pp. 1–31, 2023.

[4] A. Lakshman and P. Malik, "Cassandra: a decentralized structured storage system," *ACM SIGOPS operating systems review*, vol. 44, no. 2, pp. 35–40, 2010.

[5] A. Thomson, T. Diamond, S.-C. Weng, K. Ren, P. Shao, and D. J. Abadi, "Calvin: fast distributed transactions for partitioned database systems," in *ACM SIGMOD*, 2012, pp. 1–12.

[6] J. C. Corbett, J. Dean, M. Epstein, A. Fikes, C. Frost, J. J. Furman, S. Ghemawat, A. Gubarev, C. Heiser, P. Hochschild *et al.*, "Spanner: Google's globally distributed database," *ACM TOCS*, vol. 31, no. 3, pp. 1–22, 2013.

[7] A. Dragojević, D. Narayanan, M. Castro, and O. Hodson, "{FaRM}: Fast remote memory," in *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)*, 2014, pp. 401–414.

[8] B. Dageville, T. Cruanes, M. Zukowski, V. Antonov, A. Avanes, J. Bock, J. Claybaugh, D. Engovatov, M. Hentschel, J. Huang, A. W. Lee, A. Motivala, A. Q. Munir, S. Pelley, P. Povinec, G. Rahn, S. Triantafyllis, and P. Unterbrunner, "The snowflake elastic data warehouse," in *Proceedings of the 2016 International Conference on Management of Data*, ser. SIGMOD '16, 2016, p. 215–226.

[9] K. Ren, D. Li, and D. J. Abadi, "Slog: Serializable, low-latency, geo-replicated transactions," *VLDB*, vol. 12, no. 11, 2019.

[10] Huawei, "Huawei cloud: Everything as a service," https://www.huaweicloud.com/intl/en-us/, 2024.

[11] Amazon, "Aws global infrastructure," https://aws.amazon.com/about-aws/global-infrastructure/?nc1=h_ls, 2024.

[12] Google, "Introduction to bigquery omni," https://cloud.google.com/bigquery/docs/omni-introduction, 2024.

[13] SnowFlake, "Snowflake database," https://docs.snowflake.com/en/sql-reference/snowflake-db, 2024.

[14] Terraform, "Deploy federated multi-cloud kubernetes clusters," https://developer.hashicorp.com/terraform/tutorials/networking/multicloud-kubernetes, 2024.

[15] DatenLord, "A high-performance geo-distributed metadata management system," https://datenlord.github.io/xline-home/#/, 2024.

[16] Q. Zhuang, X. Shi, S. Liu, W. Lu, Z. Zhao, Y. Chen, T. Li, A. Pan, and X. Du, "Geotp: Latency-aware geo-distributed transaction processing in database middlewares (extended version)," *arXiv preprint arXiv:2412.01213*, 2024.

[17] M. Mohiuddin, M. Primorac, E. Stai, and J.-Y. Le Boudec, "Fcr: Fast and consistent controller-replication in software defined networking," *IEEE Access*, vol. 7, pp. 170 589–170 603, 2019.

[18] W. Li, T. Li, D. Zhang, L. Dai, and Y. Chai, "Distributed consensus algorithms for cross-domain data management: state-of-the-art, challenges and perspectives," *Big Data Research*, vol. 9, no. 4, pp. 2–15, 2023.

[19] Q. Zhang, J. Li, H. Zhao, Q. Xu, W. Lu, J. Xiao, F. Han, C. Yang, and X. Du, "Efficient distributed transaction processing in heterogeneous networks," *Proceedings of the VLDB Endowment*, vol. 16, no. 6, pp. 1372–1385, 2023.

[20] L. Li, K. Xu, T. Li, K. Zheng, C. Peng, D. Wang, X. Wang, M. Shen, and R. Mijumbi, "A measurement study on multi-path tcp with multiple cellular carriers on high speed rails," in *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, 2018, pp. 161–175.

[21] T. Li, K. Zheng, K. Xu, R. A. Jadhav, T. Xiong, K. Winstein, and K. Tan, "Tack: Improving wireless transport performance by taming acknowledgments," in *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*, 2020, pp. 15–30.

[22] L. Lamport, "Paxos made simple," *ACM SIGACT News*, pp. 51–58, 2001.

[23] D. Ongaro and J. Ousterhout, "In search of an understandable consensus algorithm," in *USENIX ATC*, 2014, pp. 305–319.

[24] J. Xu, W. Wang, Y. Zeng, Z. Yan, and H. Li, "Raft-plus: Improving raft by multi-policy based leader election with unprejudiced sorting," *Symmetry*, vol. 14, no. 6, p. 1122, 2022.

[25] S. Liu and M. Vukolić, "Leader set selection for low-latency geo-replicated state machine," *IEEE TPDS*, vol. 28, no. 7, pp. 1933–1946, 2016.

[26] F. Ryabinin, A. Gotsman, and P. Sutra, "Swiftpaxos: Fast geo-replicated state machines," in *USENIX NSDI*, 2024.

[27] Z. Xu, C. Stewart, and J. Huang, "Elastic, geo-distributed raft," in *IEEE/ACM IWQoS*, 2019.

[28] S. J. Park and J. Ousterhout, "Exploiting commutativity for practical fast replication," in *USENIX NSDI*, 2019.

[29] A. Charapko, A. Ailijiang, and M. Demirbas, "Pigpaxos: Devouring the communication bottlenecks in distributed consensus," in *ACM SIGMOD*, 2021.

[30] L. Lamport, "Fast paxos," *Springer DISC*, vol. 19, pp. 79–103, 2006.

[31] I. Moraru, D. G. Andersen, and M. Kaminsky, "There is more consensus in egalitarian parliaments," in *ACM SOSP*, 2013.

[32] F. Nawab, D. Agrawal, and A. El Abbadi, "Dpaxos: Managing data closer to users for low-latency and mobile applications," in *ACM SIGMOD*, 2018, pp. 1221–1236.

[33] R. Taft, I. Sharif, A. Matei, N. VanBenschoten, J. Lewis, T. Grieger, K. Niemi, A. Woods, A. Birzin, R. Poss *et al.*, "Cockroachdb: The resilient geo-distributed sql database," in *ACM SIGMOD*, 2020, pp. 1493–1509.

[34] W. Zhou, Q. Peng, Z. Zhang, Y. Zhang, Y. Ren, S. Li, G. Fu, Y. Cui, Q. Li, C. Wu *et al.*, "Geogauss: Strongly consistent and light-coordinated oltp for geo-replicated sql database," *ACM SIGMOD*, vol. 1, no. 1, pp. 1–27, 2023.

[35] Huawei, "Gaussdb," https://www.huaweicloud.com/intl/en-us/product/gaussdb.html, 2024.

[36] PingCAP, "Tidb," https://www.pingcap.com/tidb/, 2024.

[37] A. Cloud, "Polardb," https://www.alibabacloud.com/en/product/polardb?_p_lc=1, 2024.

[38] A. Verbitski, A. Gupta, D. Saha, M. Brahmadesam, K. Gupta, R. Mittal, S. Krishnamurthy, S. Maurice, T. Kharatishvili, and X. Bao, "Amazon aurora: Design considerations for high throughput cloud-native relational databases," in *ACM SIGMOD*, 2017, pp. 1041–1052.

[39] Neon, "Neon," https://neon.tech/, 2024.

[40] R. Neiheiser, M. Matos, and L. Rodrigues, "Kauri: Scalable bft consensus with pipelined tree-based dissemination and aggregation," in *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles*, 2021, pp. 35–48.

[41] ETCD, "Etcd-raft," https://github.com/etcd-io/etcd, 2024.

[42] E. Sakic, P. Vizarreta, and W. Kellerer, "Seer: Performance-aware leader election in single-leader consensus," *arXiv preprint arXiv:2104.01355*, 2021.

[43] T. Li, D. Xu, B. Wu, X. Guo, D. Jiang, C. Luo, W. Lu, and X. Du, "Transmission in wide-area deterministic networking: A survey," *Journal of Software*, pp. 1–30, 2024.

[44] L. Foundation, *Traffic Control (tc) and Network Emulation (netem)*, 2024, accessed: 2024-07-23. [Online]. Available: https://man7.org/linux/man-pages/man8/tc.8.html

[45] I. Amazon Web Services, *AWS Latency Monitoring*, 2024, accessed: 2024-07-23. [Online]. Available: https://aws.amazon.com/cloudwatch/

[46] F. N. Fritsch and R. E. Carlson, "Monotone piecewise cubic interpolation," *SIAM Journal on Numerical Analysis*, pp. 238–246, 1980.

[47] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears, "Benchmarking cloud serving systems with ycsb," in *ACM symposium on Cloud computing*, 2010, pp. 143–154.

[48] Huawei Cloud, "Two-site and three-center disaster recovery solution," 2024, accessed: 2024-07-22. [Online]. Available: https://support.huaweicloud.com/intl/en-us/bestpractice-sdrs/sdrs_bp_cbr_0000.html