MovR as a Benchmark for Geo-Distributed Databases

Performance Evaluation and Insights

Wilhelm Marcu

Supervisors: Asterios Katsifodimos, Oto Mráz, George Christodoulou, Kyriakos Psarakis

EEMCS, Delft University of Technology, The Netherlands

Name of the student: Wilhelm Marcu
Final project course: CSE3000 Research Project
Thesis committee: Asterios Katsifodimos, Oto Mráz, George Christodoulou, Kyriakos Psarakis, Koen Langendoen

Abstract

Distributed systems are vital for handling large-scale data and rely on geo-distributed databases to ensure low latency and high availability. Traditional benchmarks, such as TPC-C and YCSB-T, are not designed to handle the complexities of geo-distributed environments and do not allow for configuration of multi-home transaction ratios or dynamic data access patterns. To fill this gap, we implement a benchmark based on the MovR workload and assess its performance on the Detock, Janus, SLOG, and Calvin geo-distributed database systems. Key insights revealed through experiments are that network conditions act as a major bottleneck and high concurrency leads to unsustainable latency spikes which severely limits scalability.

1 Introduction

Distributed systems have become the standard solution for handling the large-scale data requirements of modern applications [5]. Unlike centralised systems, distributed systems are able to scale horizontally and are more reliable due to higher fault tolerance. In the context of data management, such systems need to use geo-distributed databases to ensure low latency and high availability for end users, whilst also complying with data regulations across different geographic regions [13]. Designing efficient geo-distributed databases is more challenging than single-region databases due to additional considerations such as optimising data locality, coordinating distributed transactions, and managing high-latency communication between regions.

To assess the performance of these more sophisticated database systems, benchmarks are used to test the system under different workloads and scenarios. Two well-known, industry standard benchmarks are TPC-C [17] and YCSB-T [3] but these are limited in scope since they were not developed for a geo-distributed setting. More specifically, these benchmarks lack mechanisms to control the ratio of distributed transactions or the number of nodes involved, which significantly impacts performance in distributed systems [14]. Additional limitations include the lack of support for adaptive hot data splitting, storage movement for load balancing, and elasticity mechanisms to handle contention-heavy or skewed workloads [14]. To combat these shortcomings, there is a need to create a new tunable benchmark that covers the additional complexities of geo-distributed environments.

This paper aims to aid this endeavour by experimentally examining the strengths, weaknesses, and blind spots of the MovR [2] benchmark across various geo-distributed database systems. More specifically, we evaluate MovR in Detock [12], Janus [11], SLOG [15], and Calvin [16]. The goal is to answer the following research question: *how effective is the MovR benchmark for evaluating the performance characteristics of geo-distributed databases in terms of throughput, latency, bytes transferred, and resource demands?* The main contributions of this work are:

- An implementation of a benchmark focusing on the MovR workload (Section 3)

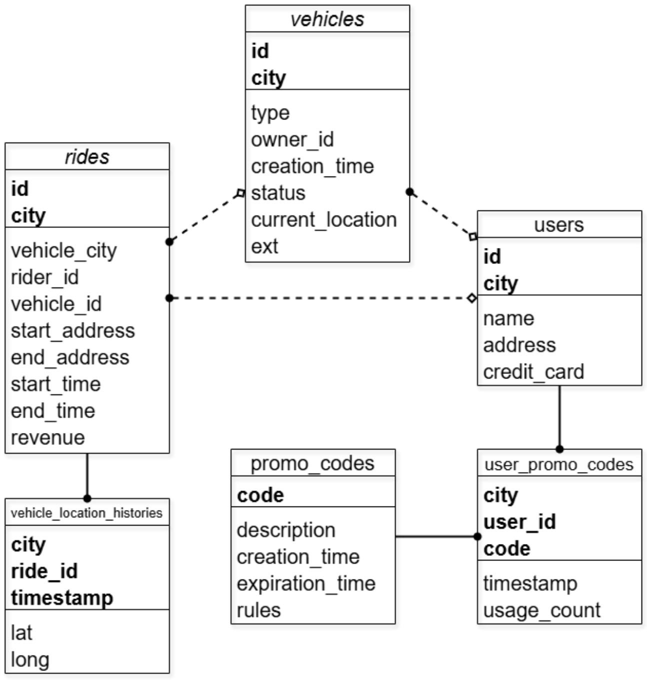


Figure 1: MovR schema

- Deployment and execution of the benchmark on four database systems under six scenarios (Section 4)
- Insights into the performance of the benchmark (Section 5)

The remainder of this paper is structured as follows: Section 2 provides background on the MovR benchmark and the database systems used to run it. Section 3 describes the implementation details of MovR. Section 4 outlines the experimental setup and results while Section 5 interprets these results and discusses ways to expand upon this work. Finally, Section 6 presents the main findings and Section 7 addresses the responsible research principles used during the study.

2 Background

2.1 MovR Workload

Traditional benchmarking techniques often focus on a single workload type such as Online Transaction Processing (OLTP) in the case of TPC-C, or key-value operations in the case of YCSB-T. Unlike these two benchmarks, MovR [2] takes a hybrid approach using a workload featuring both OLTP and time-series components, blending short-lived, ACID-compliant transactions with high-volume, append-heavy writes. MovR models a vehicle sharing service consisting of six tables as can be seen in Figure 1.

The schema's main table is *users* which experiences frequent reads but infrequent writes and updates. The *vehicles* table tracks vehicle ownership and sees frequent reads and updates since it contains fields for status and current location. The core of the workload is the *rides* table which records ride details and undergoes frequent reads, writes, and updates as ride data evolves. The *vehicle_location_histories*

Table 1: Transaction Profile

Transaction	Mix	MH/MP	Operation Intensity		
			Read	Write	Update
View Vehicles	30%	no	high	-	-
Add User	5%	no	-	low	-
Add Vehicle	5%	yes	low	low	-
Start Ride	15%	yes	high	high	low
Track Location	30%	no	-	high	-
End Ride	15%	yes	-	-	high

An overview of MovR transactions. From left to right: the chance of a transaction being generated in the workload (Mix), whether the transaction is able to involve multiple cities (MH/MP), and the operation intensity of the transaction.

table stores time-series location data during rides and is write-heavy. The last two tables manage business logic involving discount codes. Globally available codes are stored in the *promo_codes* table whereas the *user_promo_codes* table tracks codes applied by each user.

The workflow consists of six transaction types which are presented in Table 1. In order to construct a workload from this workflow, we assign a probability for each transaction to occur and sample at random to determine the next transaction produced by the workload. We derived these weights based on reasoned estimates to reflect plausible real-world behaviour and usage patterns. While prior work such as Caerus [8] simplifies MovR, using only the *Start Ride* transaction, this approach preserves fidelity to the original workload design and maintains realistic proportions. Consequently, only 35% of the transactions generated are able to be multi-partition or multi-home because only *Add Vehicle*, *Start Ride*, and *End Ride* can potentially access data across multiple cities.

2.2 Related Work

Research on geo-distributed transaction processing has produced a variety of database systems each with different trade-offs. Calvin [16] orders all transactions via a Paxos-based sequencer and replicates transactions to all replicas. This guarantees strict serializability with no throughput loss, but incurs two wide-area network (WAN) round-trips per transaction. Notably, few geo-distributed databases support strictly serializable transactions in practice because implementations typically impose severe constraints on transaction scope or latency [4]. While deterministic protocols like Calvin avoid runtime coordination, Harding et al. demonstrate they suffer from scheduling bottlenecks and reconnaissance overheads when dealing with dynamic data [7].

To improve upon this drawback, SLOG [15] takes a different approach by distinguishing between single-region and multi-region transactions. In SLOG, a home replica locally commits single-region transactions immediately, while multi-region transactions are broken into log fragments that are synchronised globally. This design yields dramatically lower tail latencies under locality but does not benefit workloads with heavy multi-home transactions. More recent work further reduces coordination such as Caerus [8] which merges partial orders from each region so that any transaction can commit after a single WAN round-trip. Another example is Detock [12] which sidesteps the global log entirely by employing a decentralised graph-based deadlock resolution algorithm. Similarly, Janus [11] uses dependency graphs to avoid global coordination but addresses a different problem: it combines transaction ordering and replica agreement into a single protocol but restricts transactions to one-shot stored procedures which limits its applicability for interactive workloads. Harding et al. remark that these graph based deadlock detection schemes suffer from "substantial network communication" overheads. [7].

The evaluation of distributed database systems requires benchmarks that capture the unique characteristics of geo-distribution. TPC-C and YCSB-T, which have already been discussed, neglect WAN-specific latency, synchronisation costs, and multi-region access patterns. Specialised benchmarks have emerged to address these gaps. GeoYCSB [9] makes use of spatial queries and data distribution across regions, but its focus on NoSQL limits applicability to transactional systems. Other examples include SmallBank [1] which is designed to evaluate systems with serializable snapshot isolation, and PeakBench [18] which models high-intensity workloads under high contention. In a comprehensive review of available transactional benchmarks, Qu et al. find that few benchmarks provide control over the "ratio of distributed transactions and the number of spanning nodes" and that none model storage movement [14].

3 Benchmark Implementation

We build our implementation on top of the existing Detock codebase [1] because Detock supports pluggable storage layers and already re-implements Calvin, SLOG, and Janus [12]. This creates a level playing field for comparing the systems fairly under identical conditions. The benchmark implementation is designed to offer flexible options for controlling the MovR workload. At its core, the *NextTransaction* function selects the next transaction to perform at random based on a configurable weighted distribution. Each transaction has its own corresponding generator which populates a Protocol Buffers object with the required data. These objects are subsequently processed by the execution layer which handles low-level storage interactions.

Multi-home and multi-partition transactions are generated probabilistically independent based on arguments passed to the workload generator. Transactions such as *AddVehicle*, *StartRide*, and *EndRide*, may span multiple regions or partitions when their generators sample remote cities. These cities are tracked using a three-dimensional array *city_index*. At the top level, cities are grouped by their assigned partitions. Within each partition, cities are further subdivided by their associated regions. Finally, each region contains a list of specific cities belonging to it. This structure is populated when the workload is initialised and cities are evenly allocated using a deterministic modulo-based mapping scheme. Table

[1] https://github.com/umd-dslam/Detock

Table 2: City Distribution

	Region 0	Region 1
Partition 0	city_0, city_4	city_2, city_6
Partition 1	city_1, city_5	city_3, city_7

Example city mapping with eight cities distributed over two partitions and two regions

2 demonstrates this mapping scheme by showing how eight cities are distributed over two partitions and two regions. The reason we chose this partitioning strategy is because all of the tables in the schema, except for *promo_codes*, have composite keys which include the city attribute. In the case of *promo_codes* which is a global table, each node in the system can safely maintain its own replica without significantly affecting performance because write and update operations for this table are not performed by the benchmark.

Skewed access patterns are implemented using Zipfian distributions which are initialised using a skew factor supplied as an argument to the workload. Transaction generators use these Zipfian distributions when generating local record IDs, where high skew values increase contention for popular records. Based on this local ID and the city index, a global 64-bit integer ID is constructed where the first 16 bits identify the city and the remaining 48 bits identify the local record. This scheme allows transactions to span multiple partitions and regions while also maintaining the desired hotspot behaviour.

In geo-distributed applications, access patterns can be used to uncover workload characteristics which can be exploited for performance gains [6]. Typically, access patterns exhibit predictable temporal shifts across regions since user activity tends to follow daytime peaks in local time-zones. These fluctuations are important to consider because database systems must dynamically rebalance the load to prevent regional overload during local peaks. Resource utilisation efficiency also depends on anticipating these access patterns which is beneficial for reducing costs. This sun-tracking phenomenon, similar to how sunflowers follow the sun, necessitates benchmarks that model cyclical workload rotations to simulate realistic hotspots.

To capture these time-dependent access patterns, our benchmark implementation supports an adjustable sunflower scenario. This feature is implemented by creating a weighted distribution used to decide the home region for each transaction, similar to how the transaction type is decided. These weights are updated before each transaction based on the time elapsed since the start of the workload. Fine-tuning of this behaviour can be achieved using three configurable parameters: *sunflower_max* sets the peak traffic percentage for the hotspot region, *sunflower_falloff* determines how sharply demand exponentially decays in neighbouring regions, and *sunflower_cycles* defines how many times the hotspot completes a full rotation across all regions. The weight w_i for region i is computed as follows:

1. Compute the cyclic position of the "sun":

$$s = \left(\frac{t_{\text{elapsed}}}{t_{\text{duration}}} \cdot c \cdot N\right) \bmod N \qquad (1)$$

   where t_{elapsed} is the time since workload start, t_{duration} is the total workload duration, c is *sunflower_cycles*, and N is the number of regions.

2. For each region $i \in [0, N-1]$, calculate the wrapped distance to the sun:

$$d_i = \min(|s-i|, N-|s-i|) \qquad (2)$$

   where s is the position of the sun, i is the region index, and N is the number of regions.

3. Compute unnormalised weight:

$$w'_i = m \cdot (1-f)^{d_i} \qquad (3)$$

   where m is *sunflower_max*, f is *sunflower_falloff*, d is distance to the sun, and i is the region index.

4. Normalise weights to sum to 100%:

$$w_i = \frac{w'_i}{\sum_{j=0}^{N-1} w'_j} \cdot 100 \qquad (4)$$

   where w' is the unnormalised weight, i is the region index, and N is the number of regions.

4 Benchmark Performance Evaluation

4.1 Experimental Setup

We deployed the MovR workload on the Detock, Janus, SLOG and Calvin systems using Docker containers on a four-node cluster. Each node ran Ubuntu 22.04.5 and featured two AMD EPYC 7H12 64-Core Processors (128 cores, 256 threads, simultaneous multithreading enabled) with 503 GiB DDR4 RAM and a 446.6 GB primary SSD. The nodes communicated via 10 Gbps Ethernet using MTU 9000 jumbo frames.

The cluster followed a two-region setup, each with two partitions, a single replica, and a client from the opposite region. The default network latency between each cluster node is presented in Table 3. We configured the MovR workload with 1,000 simulated cities using 5 millisecond sequencer batches (synchronised) and a 2,000 microsecond timestamp buffer. Other system parameters included three worker threads per node and two log managers. Before executing the workload, we pre-loaded the tables with the following number of records: 10,000 users, 5,000 vehicles, 30,000 rides, 50,000 histories, 1,000 promo codes, and 20,000 user promo codes.

We ran the experiments on each system multiple times, each time focusing on a specific scenario under which to test the workload. In total, six scenarios were considered:

- Baseline: measures system performance under varying proportions of multi-home transactions.
- Skew: evaluates resilience to non-uniform access patters by progressively concentrating requests on hot records.
- Scalability: Tests how throughput changes as the number of clients increase.

Table 3: Average round-trip time (RTT) between cluster nodes (ms)

	R0P0	R0P1	R1P0
R0P1	0.143	–	–
R1P0	0.151	0.169	–
R1P1	0.097	0.110	0.092

R0P1 indicates the node in region 0 and partition 1. The values are calculated as the average of bidirectional measurements taken over 10 pings.

- Network: Assesses performance under controlled network latency injections between nodes.
- Packet Loss: measures fault tolerance on an unreliable network with increasing packet drop rates.
- Sunflower: models dynamic workload hotspots according to time-zone patterns.

For each scenario tested, 2,000,000 transactions were generated using closed-loop clients, meaning that clients waited for a response from the previous transaction before sending a new one. For all the scenarios, the probability of multi-home and multi-partition transactions was set to 50% except for the baseline scenario, where the multi-home percentage was the independent variable. Since only 35% of the transactions are eligible to be multi-home or multi-partition, the actual percentage generated by the workload is 17.5%. To help focus the research, we devised following sub-questions to match each scenario:

1. How does throughput and latency vary with increasing proportions of multi-home transactions in the MovR workload under normal operating conditions?
2. How does varying the access skew impact throughput when running the MovR workload?
3. How does commit rate and latency scale with increasing client load when running the MovR workload?
4. How does end-to-end transaction latency change when artificial network delay is introduced during the MovR workload?
5. How does progressive packet loss affect commit rate when running the MovR workload?
6. How does introducing temporal region hotspots affect the performance of the MovR workload?

We collected the following metrics during the experiment:

- Throughput (transactions/second): measures sustained processing capacity.
- Latency (milliseconds): quantifies system responsiveness at p50, p90, and p95 percentiles.
- Bytes transferred (megabytes): measures inter-region traffic volume via per-server packet counters.
- Cost ($): estimated monetary cost based on resource utilisation patterns and computation, network and storage overheads.

The total hourly cost C_{total} is calculated as:

$$C_{\text{total}} = C_{\text{VM}} + \left(\frac{C_{\text{data}}}{T}\right) \cdot 3600 \tag{5}$$

where:

- C_{VM} is the hourly cost of all four virtual machines, estimated by scaling down the average per-hour price of an AWS m4.2xlarge instance.
- C_{data} is the total data transfer cost during the experiment, using a symmetric, uniform rate of \$0.02 per gigabyte transferred between regions.
- T is the experiment duration in seconds.

4.2 Results

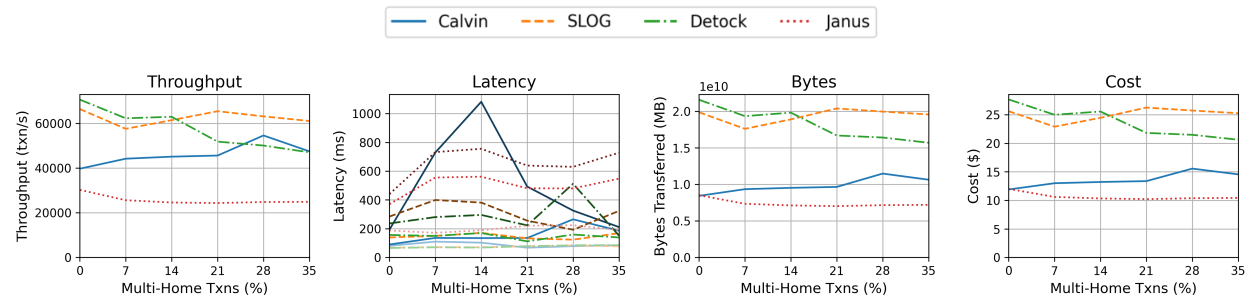
Baseline and Skew Scenarios

The experimental results for each scenario can be seen in Figure 2. In the baseline scenario (Figure 2a), SLOG demonstrates the highest throughput, maintaining approximately 60,000 txn/s across all multi-home percentages. Detock also has a high initial throughput but steadily declines as more multi-home transactions are generated. Calvin's throughput remains relatively stable at around 45,000 txn/s while Janus shows the lowest throughput, hovering around 25,000 txn/s.

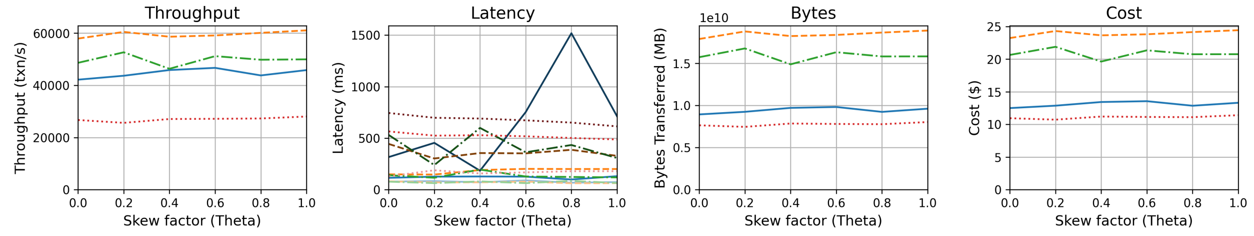
In terms of latency, Detock, SLOG, and Calvin generally maintain sub-200 millisecond latencies at both the 50^{th} and 90^{th} percentiles. In contrast, Janus has a significantly higher latency of nearly 600 milliseconds at the 90^{th} percentile, which is almost triple compared to the other systems. When measuring at the 95^{th} percentile, Calvin experiences significant spikes, even at low multi-home percentages, whereas other systems have comparably lower and more stable latencies.

The bytes transferred and cost metrics are affected by the throughput and so generally follow the same pattern for all scenarios. However, in the baseline scenario, Calvin and Detock have similar throughput for high percentages of multi-home transactions yet Calvin operates at 25% lower cost. SLOG's approach of distinguishing between single-region and multi-region transactions likely contributes to its consistently low latency and high throughput. Likewise, Detock's decentralised deadlock resolution mechanism also appears to be effective in maintaining high performance under the MovR workload, albeit under higher cost than Calvin.

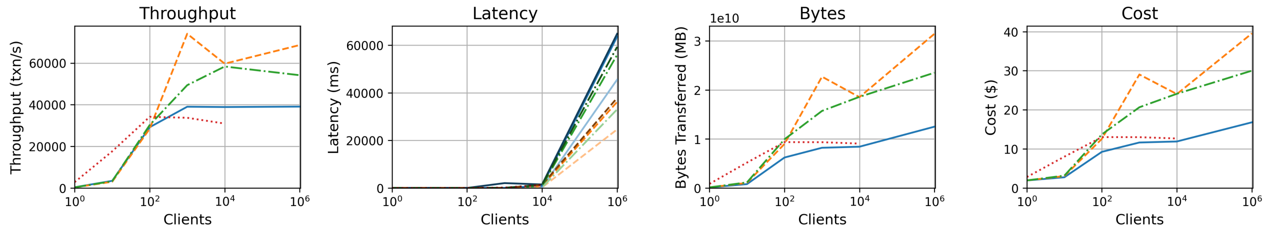
Figure 2b shows the results of the skew scenario and throughput follows the same pattern observed in the baseline scenario, with SLOG performing the best, followed by Detock, Calvin, and then Janus. An unexpected result for this scenario is that the skew factor seems to have little impact on throughput, and consequently, on bytes transferred and cost as well. Calvin still displays large latency spikes at the 95^{th} percentile and Janus still has abnormally high latency compared to the other systems. Janus consistently underperforms in throughput and latency across both baseline and skew scenarios. A possible explanation for this is that Janus uses Two-Phase Commit for cross-partition transactions which incurs a large overhead due to synchronous coordination.



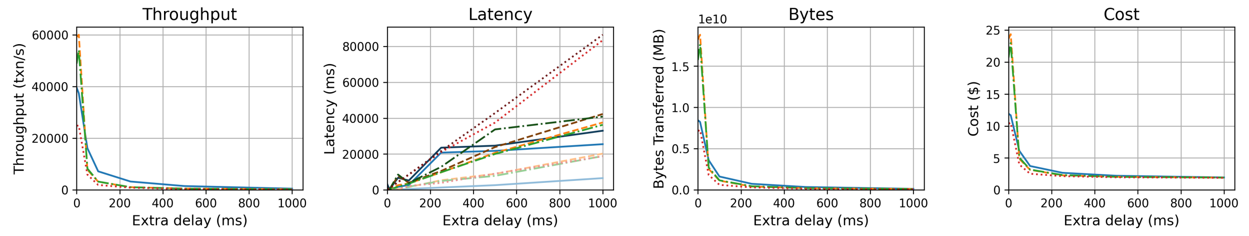
(a) Baseline scenario



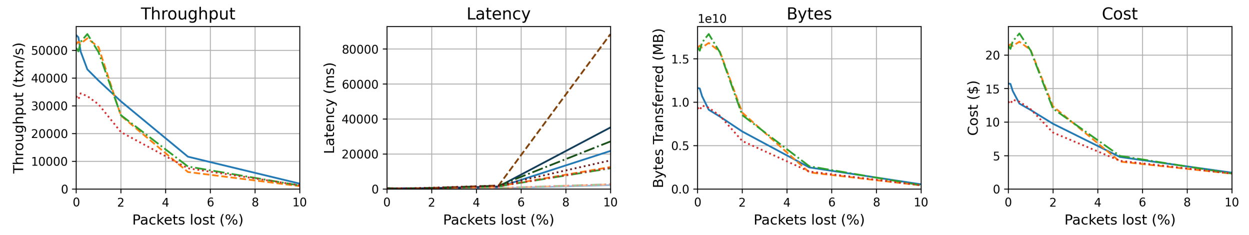
(b) Skew scenario



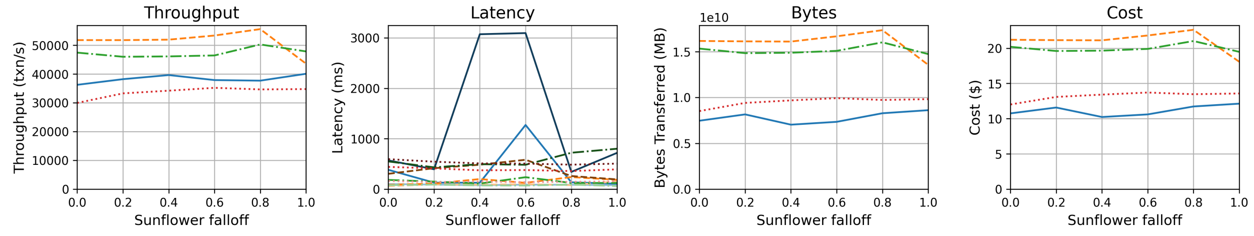
(c) Scalability scenario



(d) Network scenario



(e) Packet loss scenario



(f) Sunflower scenario

Figure 2: Complete experimental results for each scenario

Scalability and Network Scenarios

The results for the scalability scenario (Figure 2c) show that the throughput for SLOG and Detock peaks at 10^3 and 10^4 clients respectively. Calvin is able to maintain a steady throughput of 40,000 txn/s after reaching 10^3 clients up until 10^6 clients. Janus on the other hand, despite having higher initial throughput for a low number of clients, peaks at only 10^2 clients. Janus failed to complete the scalability test within a reasonable time frame after scaling to 10^4 clients, forcing us to terminate the experiment prematurely. With regard to latency, it is only noticeable after 10^2 clients and only starts to grow drastically after 10^4 clients. Latency scales the best for SLOG, followed by Detock and then Calvin.

Moving on to the network scenario, the results presented in Figure 2d show that the throughput for all systems rapidly degrades as the extra delay increases, dropping significantly even with small delays. Calvin appears to maintain the highest throughput in this scenario but all systems behave similarly after the delay exceeds 500 milliseconds, with throughput approaching 0. Latency increases at different rates for each system, with Detock and SLOG having the best performance initially. Calvin and Janus have higher initial latency, but after 250 milliseconds of extra delay, Calvin manages to stabilise to an almost constant latency of 20,000 milliseconds and even outperforms Detock and SLOG when the extra delay is greater than 600 milliseconds. Unlike Calvin, Janus is unable to stabilise and continues to grow sharply after 250 milliseconds, yielding the worst latency performance.

Packet Loss and Sunflower Scenarios

Figure 2e shows how the systems perform on unreliable networks where packet loss may occur. Even at low packet loss percentages, the throughput for all systems drops sharply reaching 10,000 txn/s at a packet loss percentage of 5%. Detock and SLOG do not appear to be affected by packet losses smaller than 1% but Calvin is able to maintain superior throughput for packet losses greater than 2%. In terms of latency, Detock, SLOG, and Janus have near identical latency at the 90^{th} percentile, with only Calvin underperforming. When considering the 95^{th} percentile, SLOG has the worst latency scaling.

Lastly, the results for the sunflower scenario can be seen in Figure 2f which mostly indicate stable throughput for all systems. The exception to this trend is SLOG which has a noticeable drop off from 0.8 to 1.0 sunflower falloff. Regarding latency, Calvin displays a large increase for sunflower falloff values of 0.4 and 0.6 while other systems are able to maintain lower and more consistent latencies. SLOG is equipped with dynamic data remastering capabilities which allows it to efficiently deal with these shifting hotspots. Moreover, SLOG is able to remap data to new home regions which explains why it is able to maintain the lowest latency compared to other systems. Calvin's static global ordering would likely struggle to adapt efficiently to these hotspots which might explain the high latency observed in this scenario.

4.3 Latency Decomposition

In order to further investigate the results from our experiments, we decompose the latency into the various components responsible for it. The latency for each of these components is visualised using heat-maps as shown in Figure 3.

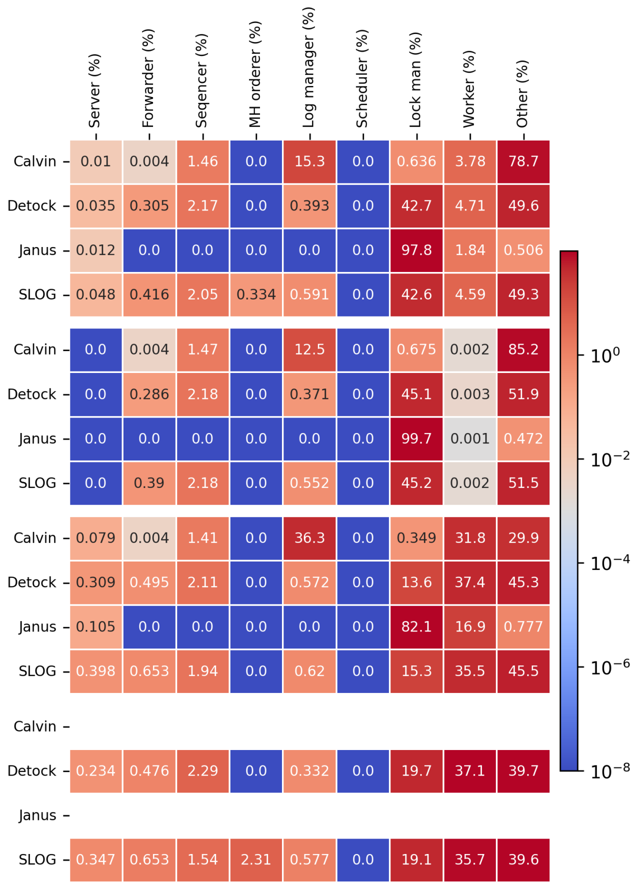


Figure 3: Latency heat-maps across various system components

The heat-maps present four distinct transaction categories: all transactions (top), single-partition single-home transactions (second), multi-partition single-home transactions (third), and multi-partition multi-home transactions (bottom). Each row represents a different database system, and the columns show the percentage of time spent in various system components during transaction processing.

Across all transaction types, the server, forwarder, sequencer, multi-home orderer, and scheduler components consume minimal processing time, typically less than 3% for all systems. Differences start to emerge when examining the log manager component. Calvin demonstrates a notable dependency on log management, spending 15.3% of its time in this component for all transactions. This behaviour becomes even more pronounced for multi-partition transactions, where Calvin allocates 36.3% of its processing time to log management. In contrast, Detock, SLOG, and Janus show minimal log manager utilisation across all transaction types, with percentages consistently below 1%.

The lock manager is a significant bottleneck for several systems, particularly Janus, which dedicates an extraordinary 97.8% of its processing time to lock management for all transactions. This aligns with the costly Two-Phase Commit protocol which Janus uses. Detock and SLOG exhibit more moderate but still significant lock manager utilisation, spending 42.7% and 42.6% respectively for all transactions. On the other hand, Calvin demonstrates minimal lock manager utilisation across all transaction types, with percentages consistently below 1%. The worker component is yet another bottleneck but only for multi-partition or multi-home transactions, where actual data processing and coordination work intensifies.

The "Other" category shown on the heat-maps accounts for substantial portions of processing time across all systems, revealing significant latency components not captured explicitly. Calvin shows the highest "Other" utilisation at 78.7% for all transactions while Detock and SLOG spend nearly half of their processing time outside of these measured components. This high percentage might be correlated with the large latency spikes associated with Calvin across multiple scenarios. Possible sources for this unaccounted for latency may include network communication overhead or garbage collection but deeper probing is required to pinpoint the exact cause.

5 Discussion

5.1 Performance Bottlenecks

The experimental results provide insights into the performance characteristics of Detock, Janus, SLOG, and Calvin under the MovR workload. While all systems initially show increased throughput with more clients, the sharp rise in latency at higher client counts suggest that coordination overheads in distributed systems eventually become a bottleneck. Detock, despite its decentralised deadlock resolution, does not appear to benefit latency when a large number of clients are involved. This might indicate that Detock has a different bottleneck such as overhead from increased complexity in graph construction or topological ordering. On the other hand, Janus' lower initial throughput and higher latency, combined with its optimisation for one-shot transactions, might mean it hits its scaling limits earlier, leading to a drastic performance drop as client count increases.

Another key observation is that even marginal increases in network delay or packet loss lead to significant degradation in throughput and a sharp increase in latency across all systems. Calvin's performance is susceptible to network delays because every transaction, even single-home ones, needs to make two WAN round-trips for its Paxos-based global ordering scheme. Similarly, SLOG needs to globally synchronise log fragments for multi-home transactions which is heavily impacted by network unreliability. Detock, while resolving deadlocks in a decentralised manner, still relies on inter-region communication which makes it vulnerable to network degradation.

Interestingly, SLOG demonstrates consistently superior performance compared to Detock across most scenarios, which differs from the original findings presented by Detock's authors [12]. Possible explanations for this performance difference can be attributed to several factors specific to the MovR workload characteristics. SLOG's architecture is particularly well-suited for workloads with strong locality patterns, as MovR exhibits with its city-based data partitioning. Furthermore, the MovR transaction mix used only allows at most 35% multi-home transactions which aligns well with SLOG's design philosophy of optimising single-region transactions while maintaining coordination capabilities for distributed operations. Additionally, SLOG's dynamic data remastering allows it to efficiently adapt to the temporal access patterns simulated by the sunflower scenario, explaining its superior performance in that specific test case.

5.2 Benchmark Effectiveness

Overall, the MovR benchmark is able to expose some of the strengths and weaknesses of different geo-distributed database systems across multiple performance dimensions, particularly in revealing the impact of network conditions and client concurrency. The benchmark reveals the performance differences between various system architectural designs through the unique performance profiles obtained for each scenario.

The benchmark contributes to the broader field of distributed systems and geo-distributed databases in several important ways. As identified by Qu et al., existing benchmarks lack mechanisms to control distributed transaction ratios and multi-region spanning characteristics [14]. This point is addressed directly through the benchmark's configuration options and scenario coverage. Furthermore, the revelation that SLOG outperforms Detock under MovR's workload characteristics, contrary to findings in the original paper [12], demonstrates that system performance is highly dependent on application patterns, business logic, and locality characteristics. This finding has significant implications for system selection in practice, indicating that architectural advantages are not universal but rather context-dependent.

Despite its strengths, the MovR benchmark has some limitations which reduces its effectiveness as a general, all-purpose benchmark for geo-distributed databases. The most significant limitation is the benchmark's inability to generate sufficient stress in the baseline and skew scenarios. This suggests that the current 35% ceiling for multi-home transactions may be insufficient to expose meaningful performance differences, and the skewing strategy requires refinement to generate more pronounced hot records. Moreover, the benchmark's focus on a single application domain (ride-sharing) limits its applicability to other application types such as social media, e-commerce, or financial services. Lastly, since the benchmark performs short, one-time experiments, it cannot capture gradual performance trends or degradation which may emerge over time in applications with constant uptime.

5.3 Future Work

For future work, several directions can be taken to expand the research further. As a first step, latency decomposition should be explored at a deeper level to reduce the high percentages of unaccounted processing times. An additional improvement is to add more geo-distributed systems to provide a more complete and diverse overview of data management strategies. Another way to improve this work is to add support for additional scenarios such as simulating a region failure to test recovery speed or simulating traffic bursts to test handling of many sudden requests.

A different avenue worth exploring is running the experiments again using clients which send at a constant rate rather than closed-loop clients which wait for a response before sending a new request. This is useful for stress testing each system to obtain results in a more realistic setting. Altering the transaction mix percentages could also be experimented with to test if the results are consistent across different transaction proportions. Lastly, alternative partitioning strategies could also be considered, such as partitioning by ID rather than city, to see if this significantly affects performance for some systems or scenarios.

6 Conclusion

This research leveraged the MovR workload to conduct a performance evaluation of four geo-distributed database systems: Detock, Janus, SLOG, and Calvin. Our experimental approach considered six different scenarios - baseline, skew, scalability, network, packet loss, and sunflower - and we collected metrics for throughput, latency, bytes transferred and cost.

Our results reveal two main insights into the performance of geo-distributed databases. The first is that network unreliability acts as a major performance bottleneck. The scenarios demonstrated that even marginal increases in network delay or packet loss lead to a drastic decrease in throughput and a sharp increase in latency across all evaluated systems. This highlights that network conditions, rather than internal database mechanics, are a significant bottleneck in geo-distributed environments. The second key insight is scalability limits and latency spikes when dealing with high concurrency. While all systems initially scaled quite well, all of them experience a dramatic performance drop in latency when dealing with 10,000 concurrent clients. This indicates that there is a fundamental challenge in maintaining low latency and consistent throughput for a large number of clients.

7 Responsible Research

This work adheres to the principles of responsible research outlined by the Netherlands Code of Conduct for Research Integrity [10]. In order to ensure reproducibility and replicability, we have documented our experimental setup in detail in Section 4, including hardware specifications, software configurations, and workload parameters. Furthermore, the source code [2] for the benchmark implementation is freely available to promote reuse and independent review. The raw data and analysis scripts are also included to facilitate reproduction of the results. We have also provided detailed instructions [3] for how to run the code and the associated scripts.

The benchmarking involved intensive experiments which put a heavy load on both system resources and network activity. Since this research was conducted in parallel with other benchmarking efforts, there was a possibility that experiments could interfere with one another. To mitigate this risk, we made sure that all experiments were performed using containers with unique port mappings. As an additional precaution, we also scheduled our experiments to run sequentially in order to avoid network congestion which would interfere with latency and byte transfer results.

We acknowledge that the MovR workload might not represent all geo-distributed application patterns. To mitigate this, we tested multiple scenarios and have been transparent about the limitations of the benchmark in Section 5. While benchmarking studies are fundamental for system development, we recognise that our results could be used to unfairly promote or criticise certain systems. We have strived for objectivity by testing under controlled conditions but we acknowledge that an alternative experimental approach might yield different results.

[2]https://github.com/delftdata/Detock/tree/movr2

[3]https://github.com/delftdata/Detock/blob/movr2/examples/movr/README.md

References

[1] Michael J. Cahill, Uwe Röhm, and Alan D. Fekete. Serializable Isolation for Snapshot Databases. *ACM Trans. Database Syst.*, 34(4):20:1–20:42, December 2009.

[2] Cockroach Labs. MovR. https://www.cockroachlabs.com/docs/stable/movr, 2025. Accessed: 2025-04-21.

[3] Akon Dey, Alan Fekete, Raghunath Nambiar, and Uwe Rohm. YCSB+T: Benchmarking web-scale transactional databases . In *2014 IEEE 30th International Conference on Data Engineering Workshops (ICDEW)*, pages 223–230, Los Alamitos, CA, USA, April 2014. IEEE Computer Society.

[4] Tamer Eldeeb, Philip A Bernstein, Asaf Cidon, and Junfeng Yang. Chablis: Fast and General Transactions in Geo-Distributed Systems. 2024.

[5] Tamer Z. Emara, Thanh Trinh, and Joshua Zhexue Huang. Geographically distributed data management to support large-scale data analysis. *Scientific Reports*, 13(1):17783, October 2023. Publisher: Nature Publishing Group.

[6] Brad Glasbergen, Michael Abebe, Khuzaima Daudjee, Scott Foggo, and Anil Pacaci. Apollo: Learning Query Correlations for Predictive Caching in Geo-Distributed Systems. 2018.

[7] Rachael Harding, Dana Van Aken, Andrew Pavlo, and Michael Stonebraker. An evaluation of distributed concurrency control. *Proceedings of the VLDB Endowment*, 10(5):553–564, January 2017.

[8] Joshua Hildred, Michael Abebe, and Khuzaima Daudjee. Caerus: Low-Latency Distributed Transactions for Geo-Replicated Systems. *Proceedings of the VLDB Endowment*, 17(3):469–482, November 2023.

[9] Suneuy Kim, Yvonne Hoang, Tsz Ting Yu, and Yuvraj Singh Kanwar. GeoYCSB: A Benchmark Framework for the Performance and Scalability Evaluation of Geospatial NoSQL Databases. *Big Data Research*, 31:100368, February 2023.

[10] KNAW, NFU, NWO, TO2-Federatie, Vereniging Hogescholen, and VSNU. Nederlandse gedragscode wetenschappelijke integriteit, 2018.

[11] Shuai Mu, Lamont Nelson, Wyatt Lloyd, and Jinyang Li. Consolidating concurrency control and consensus for commits under conflicts. In *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation*, OSDI'16, page 517–532, USA, 2016. USENIX Association.

[12] Cuong D. T. Nguyen, Johann K. Miller, and Daniel J. Abadi. Detock: High performance multi-region transactions at scale. *Proceedings of the ACM on Management of Data*, 1(2), June 2023.

[13] Qifan Pu, Ganesh Ananthanarayanan, Peter Bodik, Srikanth Kandula, Aditya Akella, Paramvir Bahl, and Ion Stoica. Low latency geo-distributed data analytics. *SIGCOMM Comput. Commun. Rev.*, 45(4):421–434, August 2015.

[14] Luyi Qu, Qingshuai Wang, Ting Chen, Keqiang Li, Rong Zhang, Xuan Zhou, Quanqing Xu, Zhifeng Yang, Chuanhui Yang, Weining Qian, and Aoying Zhou. Are current benchmarks adequate to evaluate distributed transactional databases? *BenchCouncil Transactions on Benchmarks, Standards and Evaluations*, 2(1):100031, March 2022.

[15] Kun Ren, Dennis Li, and Daniel J. Abadi. SLOG: serializable, low-latency, geo-replicated transactions. *Proceedings of the VLDB Endowment*, 12(11):1747–1761, July 2019.

[16] Alexander Thomson, Thaddeus Diamond, Shu-Chun Weng, Kun Ren, Philip Shao, and Daniel J. Abadi. Calvin: fast distributed transactions for partitioned database systems. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, pages 1–12, Scottsdale Arizona USA, May 2012. ACM.

[17] Transaction Processing Performance Council. TPC Benchmark C Standard Specification, Revision 5.11. Technical report, Transaction Processing Performance Council, February 2010. Accessed: 2025-04-21.

[18] Chunxi Zhang, Yuming Li, Rong Zhang, Weining Qian, and Aoying Zhou. Benchmarking for Transaction Processing Database Systems in Big Data Era. In Chen Zheng and Jianfeng Zhan, editors, *Benchmarking, Measuring, and Optimizing*, pages 147–158, Cham, 2019. Springer International Publishing.