TÉCNICO LISBOA

Fundação para a Ciência e a Tecnologia

Design of Geo-Distributed Shared Logs to Support Partially-Replicated Transactional Systems

Inês Martins Calado de Sousa Cardeira

Thesis to obtain the Master of Science Degree in

Computer Science and Engineering

Supervisor: Prof. Luís Eduardo Teixeira Rodrigues

Examination Committee

Chairperson: Prof. Paolo Romano
Supervisor: Prof. Luís Eduardo Teixeira Rodrigues
Member of the Committee: Prof. João Carlos Antunes Leitão

October 2025

Declaration
I declare that this document is an original work of my own authorship and that it fulfills all the requirements of the Code of Conduct and Good Practices of the Universidade de Lisboa.

Acknowledgments

First, I would like to express my sincere gratitude to my supervisor, Prof. Luís Rodrigues, for his insight, support, and steady guidance throughout this work. I am also grateful to Rafael Soares for his help and for the thoughtful discussions that shaped and strengthened this thesis.

I am profoundly grateful to my parents for their enduring support and the many sacrifices that opened doors throughout my academic life. You instilled in me resilience, integrity, and a love of learning; your encouragement—often offered quietly, consistently, and at exactly the right moments—helped shape me into the person I am today. This accomplishment is as much yours as it is mine.

To my friends, thank you for the steady support, the sanity-saving breaks, and for sharing the highs and lows together. Showing up for each other made this process lighter and better.

Finally, to my boyfriend, thank you for your support and companionship, for lifting me up whenever I stumbled, and for cheering me on when I needed it most.

To each and every one of you – Thank you.

This work was supported by national funds through Fundação para a Ciência e a Tecnologia, I.P. (FCT) under projects UID/50021/2025 and UID/PRR/50021/2025 and GLOG (financed by the OE with ref. LISBOA2030-FEDER-00771200).

Abstract

Partial replication enables scalability provided it can be supported with genuine ordering protocols, i.e., protocols in which only the nodes involved in a transaction need to participate. The coordinator-based Skeen ordering algorithm is genuine and uses a linear number of messages, but its performance depends on the coordinator's location. In this work we evaluate the benefits of informed coordinator selection in geo-distributed transactional systems. We present an experimental evaluation of these techniques to assess their advantages over uninformed alternatives.

Keywords

Distributed Transactions; Partial Replication; Distributed Logs; Genuine Total Order.

Resumo

A replicação parcial permite oferecer capacidade de escala desde que possa ser oferecida usando protocolos de ordenação genuínos, isto é, nos quais apenas os nós envolvidos numa transacção necessitam de participar. O algoritmo de ordenação de Skeen baseado em coordenador é genuíno e usa um número linear de mensagens, mas o seu desempenho é afetado pela localização do coordenador. Neste trabalho avaliamos as vantagens de selecionar de forma informada o conjunto de coordenadores em sistemas transacionais geo-distribuídos. Apresentamos uma avaliação experimental destas técnicas, para aferir as suas vantagens em relação a algoritmos não informados.

Palavras Chave

Transações Distribuídas; Replicação Parcial; Registos Distribuídos; Ordem Total Genuína.

Contents

List of Figures

List of Tables

Acronyms

CC	Cloud Computing
RTT	Round-Trip Time
DAG	Directed Acyclic Graph
LC	Lamport Clock
VC	Vector Clock
PR	Partial Replication
MR	Monotonic Reads
MW	Monotonic Writes
RYW	Read-Your-Writes
WFR	Writes-Follow-Reads
1SR	Serializability
Strong 1SR	Strict Serializability
CC	Causal Consistency
CC+	Causal+ Consistency
TCC	Transaction Causal Consistency

1

Introduction

Contents

Modern storage systems are increasingly geo-distributed and partially replicated: data is sharded across regions and each shard is replicated only on a subset of regions. Many applications require transactional guarantees (e.g., strict serializability), which in turn require a total order on the transactions that touch the same data. There are two main approaches to enforce total order in this context. Sequencer-based approaches enforce a total order via a designated site, keeping message complexity low but forcing every multi-region transaction to traverse (possibly) long network paths. Genuine ordering approaches—e.g., Skeen's algorithm [1]—coordinate only the regions that participate in a transaction, preserving locality but adding an extra communication step, which ultimately increases latency and makes coordinator placement a first-order determinant of performance.

Our central question is whether an informed choice of coordinator can reduce end-to-end latency

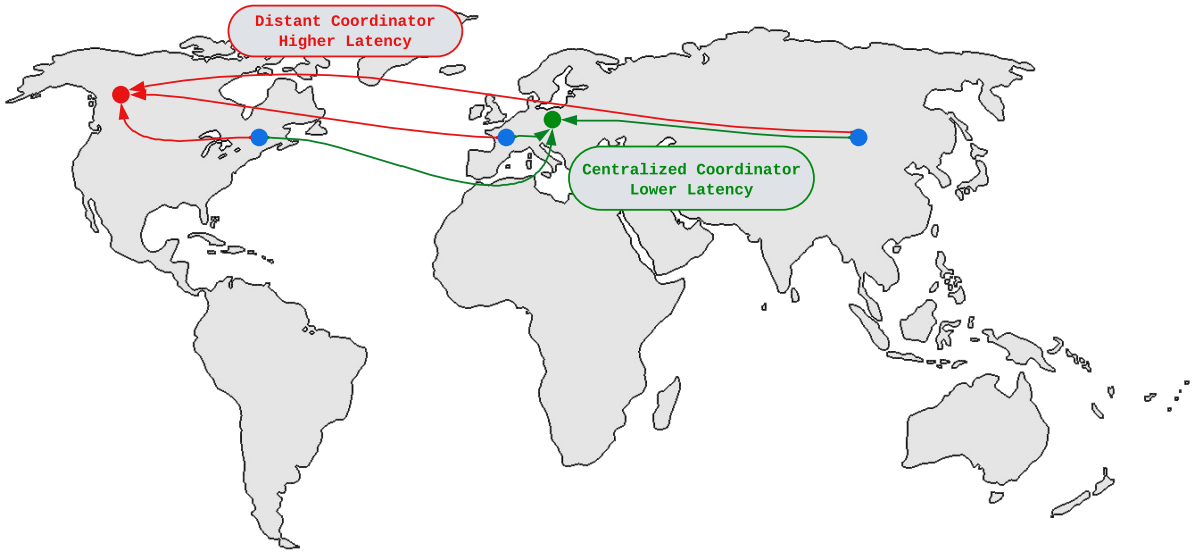

Figure 1.1: Impact of coordinator placement on end-to-end latency: a distant coordinator (red) amplifies inter-continental RTTs; an RTT-central coordinator (green) shortens the coordination path.

for multi-home transactions. We study this question in a partially replicated, geo-distributed setting and contrast it with a centralized baseline.

1.1 Motivation

Achieving total order across distant regions is challenging because the inter-continental Round-Trip Time (RTT) dominate end-to-end latency. Deployment choices (e.g., where a coordinator runs) directly change which RTTs lie on the critical path: a coordinator placed near the participants keeps both the proposal and decision exchanges between short, local links; a distant coordinator forces those same exchanges over the longest inter-continental links.

There are two main approaches to establishing a total order in geo-distributed settings. Sequencer-based designs route every multi-region transaction through a fixed site, keeping message complexity low but making latency proportional to the distance to that site, even when all participants are geographically close to each other. Genuine ordering protocols, such as Skeen's algorithm [1], coordinate *only* the regions that participate in a given transaction, preserving locality; however, they add one extra communication step and therefore become very sensitive to *where* the coordinator for that transaction is placed.

This effect is amplified in partially replicated systems. A multi-home transaction that spans nearby regions can be fast if the coordinator is RTT-central with respect to the participants, but it can become unnecessarily slow if the coordinator is located far away. Figure 1.1 illustrates the contrast: a distant coordinator (red) stretches both phases of coordination across long-haul links; a central coordinator

(green) shortens those paths and reduces total latency. Because coordinators are logical roles that can be placed at different home regions, well thought placement offers a simple lever to exploit data locality without reverting to a single global sequencer.

This dissertation addresses the following question: given a geo-distributed, partially replicated key-value store, to what extent can the informed placement of the coordinator—relying solely on topology information (RTTs) and observed locality- can lower the latency of genuine total ordering without introducing a sequencer-like bottleneck? We study this question empirically by implementing an informed variant of Skeen's protocol and compare it against a centralized baseline under controlled topologies and locality mixes, using the map in Figure 1.1 to ground the intuition behind our placement choices.

1.2 Contributions

The thesis makes the following contribution:

- We propose an architecture that applies Skeen's genuine protocol to multi-home transactions while preserving linear message complexity. We provide a practical physical placement principle—place each coordinator at an RTT-central home region among its participants (for pairs, one of the two homes) while avoiding sequencer-like hotspots.

1.3 Results

This dissertation has produced the following results:

- An implementation of an informed, coordinator-based, variant of Skeen's protocol integrated into a common C++ framework that also runs a centralized (SLOG-style [2]) baseline under identical conditions.
- An experimental evaluation on a geo-distributed testbed with artificial RTTs and different workloads, assessing end-to-end latency and per-continent behavior.

1.4 Research History

This work was carried out at INESC-ID as part of an ongoing research effort on geo-distributed, partially replicated transactional systems. In my work I have benefited from the collaboration with Rafael Soares, a PhD student working in this field.

Parts of the work described on this dissertation have been published as:

- I. Cardeira, R. Soares and L. Rodrigues. Seleção de Coordenadores em Sistemas Transaccionais Geo-Distribuídos. In Actas do décimo sexto Simpósio de Informática (Inforum), Évora, Portugal, September 2025.

This work was supported by national funds through Fundação para a Ciência e a Tecnologia, I.P. (FCT) under projects UID/50021/2025 and UID/PRR/50021/2025 and GLOG (financed by the OE with ref. LISBOA2030-FEDER-00771200).

1.5 Structure of the Document

The dissertation is organized as follows: Chapter 2 presents background on shared logs, ordering techniques, transactions and consistency models; Chapter 3 surveys related work on shared (totally ordered) logs, tree-based architectures, and locality in distributed logs, with a comparative discussion; Chapter 4 details our system architecture—model and assumptions, the coordinator-based design for total order, coordinator placement, and transaction processing; Chapter 5 reports the evaluation: setup, intra- vs. inter-continental locality, visualization and isolation of the convoy effect. Chapter 6 concludes and outlines future work.

2

Background

Contents

Understanding distributed systems requires familiarity with some key concepts that establish their design and operation. This chapter provides an overview of foundational concepts, including shared logs, log ordering techniques, transactions, consistency models, and metadata management. Each of these topics plays a critical role in maintaining the reliability, performance, and consistency of distributed systems.

2.1 Shared Log

A shared log is a distributed data structure that maintains a history of records, typically in a fault-tolerant manner. For this purpose, each record is stored in multiple storage nodes. In addition, log entries are immutable, and the only way to update the log is by adding new entries. This allows applications that feed from the log to keep a mutually consistent state. For instance, a log can be used to store commands to be executed by a state-machine: in this case, clients add commands to the log and state-machine replication can be trivially achieved by having different server replicas read the log and process commands in the order specified by the log. Shared logs have been used to support many distributed applications, including cloud service providers and other large-scale architectures, offering fault tolerance, state replication, and simplified system reconfiguration.

2.2 Log Ordering Techniques

Given that a log is replicated for fault-tolerance, replicas need to agree on the order by which entries are added to the log. This requires the execution of some form of consensus protocol with a dual purpose: replicas must agree on the set of entries that are added to the log and on the sequence number assigned to each entry. Different techniques have been proposed to assign sequence numbers to replicas, including:

Sequencers: This technique uses a centralized component, named a *sequencer*, that is responsible for ordering the operations of the system. Examples of systems that use sequencers are CORFU [3], vCorfu [4], and SLOG [2]. Sequencers avoid the costs of coordination that may occur when different replicas concurrently attempt to assign the same sequence number to different entries but may become a bottleneck in the system. Additionally, in geo-distributed settings, using a centralized sequencer may add a significant latency to the ordering procedure, since remote locations may observe large network delays when contacting the sequencer. **Figure 2.1** illustrates this penalty: even when all participants are close to each other, a distant sequencer forces long inter-continental paths.

Stabilization: In this technique, a single replica assigns to each entry a local sequence number. Different replicas can assign local sequence numbers to different entries concurrently and without coordination. Then, local sequencer numbers are exchanged among replicas and deterministically merged to establish a global total order. To perform the merge, a replica must have received local sequence numbers up to a given point from *all* other replicas. Thus, if a replica has not locally assigned sequence numbers, it needs to notify all other replicas by sending special *null* message such that deterministic merge can be applied. When using stabilization, there is a trade-off between the overhead caused by null messages and the latency of deterministic merge.

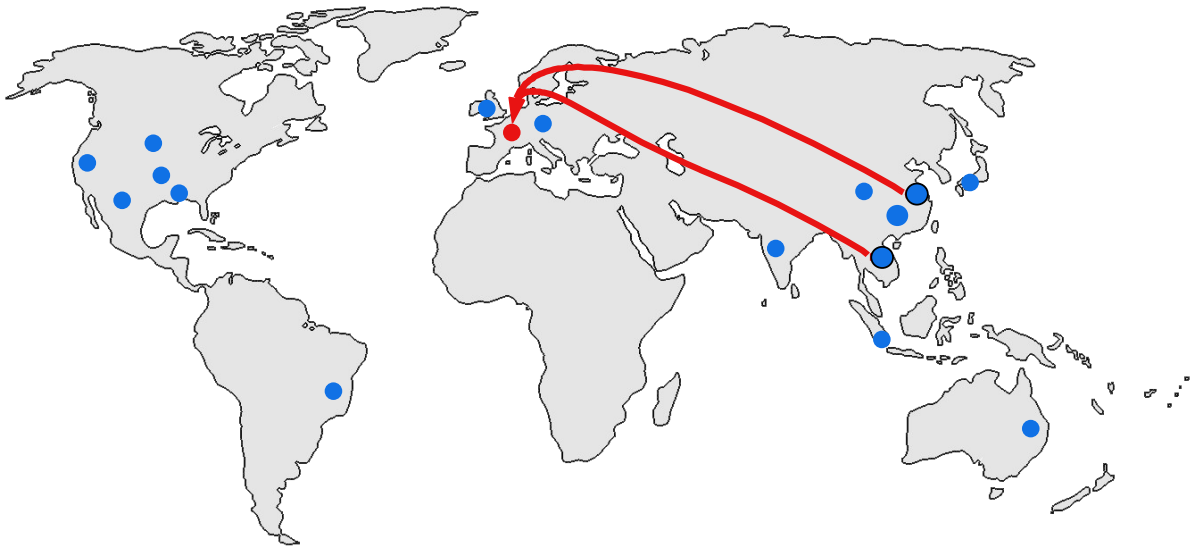

Figure 2.1: Sequencer-based ordering. A distant sequencer (red) inflates the critical path: participants that are geographically close still traverse long inter-continental links to obtain a total order.

Tree-Based Aggregation: This technique combines features of the stabilization and sequencer approaches. Similarly to stabilization based systems, different replicas can assign local sequence numbers to different entries concurrently and without coordination. These local sequence numbers are exchanged using a tree of nodes that act as message forwarders and aggregators, as they perform a deterministic merge of multiple sources before propagating a merged stream of sequence numbers. When the root node deterministically merges the streams coming from its children, it established the final total order to be used across all replicas. Some examples of systems that use such technique are Saturn [5] and ENGAGE [6].

Genuine: In genuine ordering [7], only the regions that participate in a transaction exchange messages to establish its position in the total order, preserving locality and avoiding uninvolved sites. Figure 2.2 highlights this: coordination is confined to the participating regions (green), so latency reflects nearby regions rather than a distant hub. A common genuine protocol is Skeen's algorithm [1].

2.3 Skeen's Algorithm

Skeen's algorithm [1] is a genuine total-order protocol: to order transactions, only the regions that participate in that transaction exchange messages. Each participant maintains a Lamport logical clock [8]. Ordering proceeds in two phases:

1. **Propose/Prepare:** each participating region assigns a local timestamp strictly larger than any previously issued and sends this proposal;

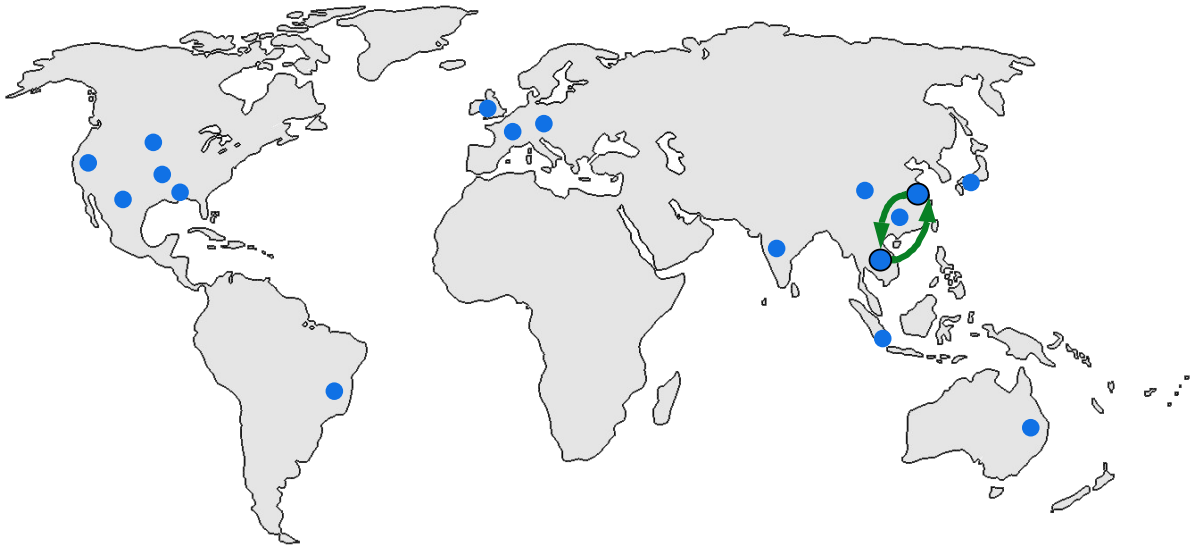

Figure 2.2: Genuine ordering. Only the participating regions (circled) coordinate; traffic remains local and the critical path is bounded by nearby RTTs.

2. **Decide:** the global timestamp is the *maximum* of the proposals and fixes the transaction's position in the total order at every participant. Upon receiving it, each region may advance its logical clock (if needed) and enqueue/commit the transaction so that it interleaves consistently with local work.

Figure 2.3 illustrates this for two regions (A and B) with a coordinator for the pair (AB): the regions propose local timestamps, the coordinator chooses the maximum (MAX), and the decided value is sent back so both regions log/execute the transaction consistently.

2.3.1 Deployment variants

There are several practical ways to exchange and aggregate proposals:

- **All-to-all** - Every participant sends its proposal to all the others and learns the maximum directly. This preserves genuineness but incurs a *quadratic* number of messages as the number of participants grows, with multiple round-trip exchanges.
- **Client aggregator** - The client collects proposals and disseminates the decided maximum (e.g., [9]). This uses a *linear* number of messages, but latency can increase if the client is far from the regions, and fault tolerance is more complex.
- **Coordinator** - A designated process—typically in one of the participating regions—aggregates proposals and returns the maximum. This also uses a *linear* number of messages while keeping the protocol genuine; however, end-to-end latency becomes highly sensitive to *where* the coordinator is placed, a central theme in our architecture and evaluation.

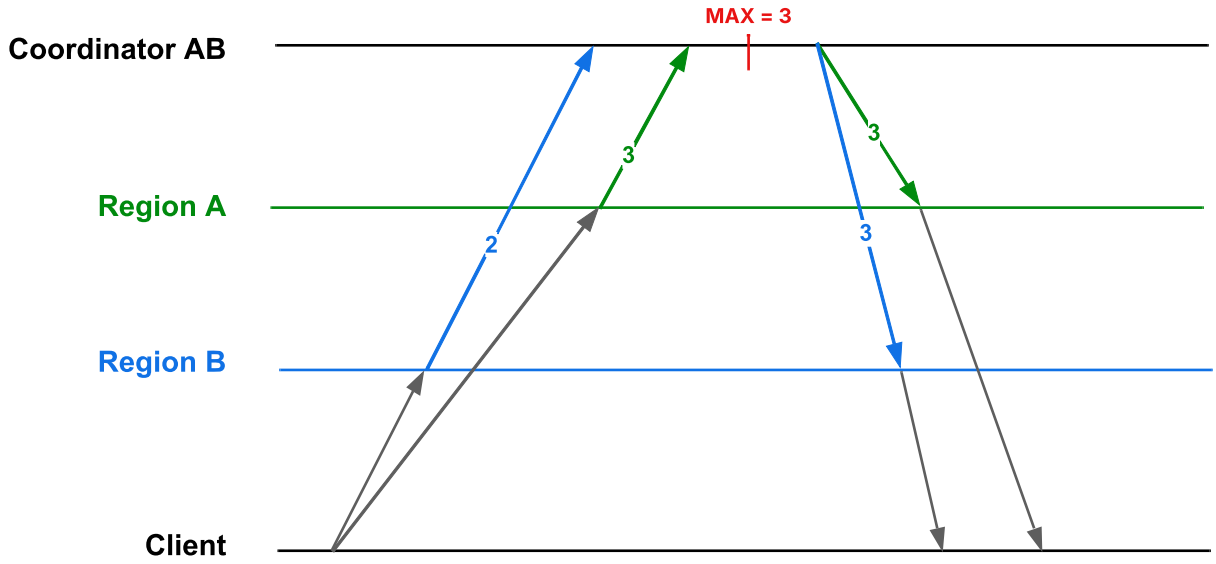

Figure 2.3: Skeen with two regions (A, B) using a pair coordinator (AB). Each region proposes a local timestamp (blue/green arrows); the coordinator decides MAX and disseminates it downward. The decided timestamp establishes the same total order at both regions.

2.4 Transactions

A transaction is a set of operations that are combined and executed as one cohesive atomic unit. Systems that support transactions often guarantee a set of properties, known as the ACID properties, namely Atomicity, Consistency, Isolation, Durability. In a distributed system, the execution of a transaction may involve multiple nodes that need to coordinate to ensure these properties. Two-phase or three-phase commit protocols are coordination protocols that can be used to ensure that all participants in a transaction agree on the outcome of the transaction, i.e., if the transaction commits or aborts, such that atomicity is preserved. Isolation is provided by the execution of some concurrency control mechanism. Many concurrency control protocols require transactions to be serialized and, therefore, require a total order to be defined. In this context, distributed shared logs can be a useful tool to serialize transactions and agree on their outcome.

2.5 Consistency Models

Consistency models have been defined for scenarios where clients submit isolated operations and for scenarios where clients submit transactions. Various consistency models manage the balance between performance and availability, each with its own trade-offs. Informally, a consistency model is denoted to be “strong” if the outcome of the concurrent execution of multiple transactions is equivalent to the execution of those transactions in isolation, in serial order, in a single replica. As it will be discussed

next, strong consistency requires operations to be totally ordered, and total order protocols may block in the presence of network partitions. This fact has been captured by the CAP theorem [10], which says that no distributed system can achieve at the same time (Strong) Consistency, Availability and Partition Tolerance.

Many consistency models ensure that operations are executed in an order that respects causality. Two operations are causally related if one operation influences or depends on the other. This relationship is captured by Lamport's "happened-before relation" [8]. For example, a write operation and a subsequent read of the same data are considered causally related.

In the following, we refer some key Non-Transactional and Transactional consistency models.

2.5.1 Non-Transactional Guarantees

Session Guarantees: [11] Set of consistency properties defined for weakly replicated systems. A session establishes a relationship between past operations performed by the client and the constraints applied to their future reads and writes, with the specific behavior depending on the set of guarantees the system provides within the session. Four guarantees have been defined, namely: *Read-Your-Writes (RYW)*, that ensures that a client observes the effects of their own writes; *Monotonic Reads (MR)*, which guarantees that once a client reads a value for a given object, they will not observe an older version of that same object in future read operations; *Monotonic Writes (MW)*, that guarantees that all updates are applied in the order they have been submitted; and *Writes-Follow-Reads (WFR)*, that ensures that an update is applied after all updates that have been previously observed by the client.

Causal Consistency (CC): Based on Lamport's "happened-before relation" [8], it dictates that all replicas observe updates in an order that is consistent with causality, while allowing concurrent events to be observed in different orders.

Causal+ Consistency (CC+): [12] It extends CC, in this context, conflicts arise when different replicas concurrently update the same object, creating multiple possible versions. CC+ ensures that all replicas independently and deterministically resolve conflicts in the same way, reaching the same state.

2.5.2 Transactional Guarantees

Transaction Causal Consistency (TCC): It extends the concept of CC by incorporating transactional semantics and ensuring atomic visibility of transactions. It guarantees that transactions operate on causally consistent snapshots, where all causal dependencies of the accessed objects are respected. Furthermore, TCC establishes causal relations between transactions, determining their order based on the causal dependencies of the objects they access.

Serializability (1SR): It ensures that the outcome of executing transactions in a distributed system is equivalent to an execution where all transactions are ordered sequentially, not necessarily matching the chronological order of transactions. In replicated systems, this property is also defined as **One-Copy Serializability**, ensuring that all replicas agree on the same global transaction order, producing results equivalent to executing all transactions sequentially on a single logical copy of the data.

Strict Serializability (Strong 1SR): It guarantees that the order of transactions across all the replicas of the system, reflecting their real-time order of execution. For this reason, since it imposes stricter constraints, it might lead to a worse throughput.

2.6 Metadata to Track Causal Dependencies

In distributed systems, metadata plays a crucial role in maintaining consistency by tracking dependencies between operations. It ensures that updates are applied in the correct order across replicas, respecting causal relationships. The choice of metadata design introduces trade-offs between storage overhead, computational complexity, and the system's ability to minimize false dependencies [5]. False dependencies occur when independent operations are incorrectly treated as dependent, causing unnecessary delays in making updates visible.

Below, we explore two common types of metadata used in distributed systems, Lamport's clocks and vector clocks:

Lamport Clock (LC)s: LCs [8] are scalars that are used to keep track of causality. The rules to increment LCs ensure that, if an event e_1 is in the causal past of another event e_2, then the LC of e_1 is guaranteed to be smaller than the LC of e_2. However, it is also possible that the clock of e_2 is larger than the clock of e_1 without e_1 being in the causal past of e_2. This is called a false dependency, that results from the fact that it is impossible to determine if two events are concurrent from scalar clocks alone. False dependencies may introduce unnecessary latencies when making updates visible.

Vector Clock (VC)s: A form of logical clocks that uses multiple entries, usually one for each process or replica in the system, with each one of them maintaining its own VC. By precisely capturing dependencies, they minimize false dependencies, enabling operations to proceed independently when no causal relationship exists, which leads to a lower visibility latency as updates can be applied without unnecessary serialization. However, this approach comes with its own disadvantages. For example, the fact that it can lead to higher storage and computation requirements, making it ill suited for systems that prioritize high throughput.

Summary

This chapter introduced the foundations for the thesis: geo-distribution and partial replication, shared logs and total order, and the main ordering techniques (sequencers, stabilization, and tree-based aggregation). We also explained genuine ordering and Skeen's algorithm, highlighting that only participating regions coordinate but that latency becomes sensitive to coordinator placement. These notions establish the vocabulary and design constraints used in the remainder of the work.

3

Related Work

Contents

In this chapter, we explore prior approaches to transaction processing and data management in distributed systems by focusing on three key areas: shared log architectures, tree-based architectures and locality in distributed logs. These systems address the challenges of consistency, scalability and efficiency in distributed environments. Each subsection highlights the design principles, trade-offs, and innovations.

3.1 Shared (Totally Ordered) Logs

We start by presenting totally ordered shared log systems, where every operation in the log is assigned a unique position in a global sequence, imposing a strict, deterministic order of all operations.

3.1.1 CORFU

CORFU [3] is a totally ordered distributed shared log implementation that uses a storage system composed of multiple SSDs, where each SSD is responsible for storing a different portion of the log. Therefore, different clients can concurrently read or write in different portions of the log without interfering with each other. Furthermore, every portion of the log is replicated in multiple SSDs that are organized in a chain. Which SSDs keep each portion of the log is defined by a centralized component in what is called a *projection*. This centralized component ensures that all participants use the same projection. Individual SSDs are augmented with a hardware mechanism that ensures that each entry of the log can only be written once. Therefore, if two clients concurrently attempt to write on the same entry, only one of the clients will succeed.

Writing in the log consists of two phases: first the client must find the most recent unused entry in the log (i.e., the head of the log) and then the client must write in all replicas of that entry. To find the log head, the client may simply parse the log until it finds the first unused entry. Unfortunately, if multiple clients do this concurrently, they will all compete for the same entry, only one of them will succeed, and the rest must repeat the process. To avoid having multiple clients competing for the same entry, CORFU uses a centralized sequencer that is responsible for assigning log entries to clients. Therefore, a client first contacts the sequencer and then performs the append operation. Updates are performed by writing in all replicas, in sequence, starting at the head of the chain. The update is complete once the tail replica is written. If a client fails after being assigned a log entry, but before it writes to all replicas, a special procedure needs to be executed to “fill” unused or partially written entries.

By leveraging the client-driven chain replication protocol and a centralized sequencer for log position assignment, CORFU achieves high throughput, enabling clients to append at the maximum rate allowed by the sequencer’s position assignment speed.

3.1.2 vCorfu

vCorfu [4] is a cloud-scale object store built over a shared log abstraction, designed to keep the advantages of systems like CORFU, such as strong consistency, while addressing their inherent limitations, including restricted read performance, through the use of materialized streams. A common issue in traditional shared log systems is the playback bottleneck, where clients must replay all updates in the log - including those irrelevant to their requests - to retrieve specific data. This often forces clients to contact multiple nodes, increasing latency. To solve these issues, vCorfu leverages materialized streams, which organize updates into separate streams, each corresponding to a specific object. These streams act as logical partitions of the global log, containing only updates relevant to their associated object. This allows clients to directly access relevant updates without traversing the global log, significantly simplifying

read operations.

Similarly to CORFU, vCorfu is composed of a sequencer and a layout layer that maps log and stream addresses to the corresponding replicas, providing clients with a global view of the system. The sequencer extends CORFU's functionality by tracking the tails of both the global log and individual streams, maintaining entry counts for each stream, ensuring a total order of updates. When a client appends data, the sequencer assigns global and stream-specific addresses. Using these addresses, the client writes updates directly to the respective log and stream replicas. Once the updates have been successfully written, a commit message is broadcast to all accessed replicas, which guarantees that stream replicas are free of gaps, enabling efficient data access. This commit process is specific to the log operations, ensuring durability and consistency at the log level.

vCorfu also supports optimistic transaction execution on clients' in-memory views. These views allow the client to cache the latest state of the objects it interacts with. During a transaction, the client tracks the versions of accessed objects - its read set - and the modifications made to objects - its write set. The sequencer, acting as a lightweight transaction manager, validates transactions by ensuring that the read set remains unchanged. If validated, the sequencer issues new tokens, allowing the client to commit updates atomically; otherwise, the transaction is rejected. This mechanism ensures snapshot isolation and prevents unnecessary writes to the log, optimizing system performance.

The introduction of materialized streams enables vCorfu to achieve better read scalability compared to traditional shared log systems, where clients must replay the global log for updates. By allowing direct reads from stream replicas, clients can retrieve individual updates or entire streams with minimal latency. However, while the additional commit messages improve read efficiency, they introduce some overhead to write operations, which can reduce throughput in write-heavy scenarios. Nonetheless, this trade-off is justified by the significant gains in read performance and scalability, particularly for workloads with frequent reads or large data sets.

3.1.3 Scalog

Scalog [13], just like CORFU, is a totally ordered shared log abstraction. While CORFU requires clients to obtain a global sequence number directly from a centralized sequencer, Scalog moves this responsibility onto the shards themselves, allowing for further batching opportunities, reducing the load onto the centralized sequencer.

Clients in Scalog can write directly to the storage server of their choice without knowing the record's global sequence number beforehand. Upon receiving a record, the server stores it locally and begins the replication process within its shard. Scalog's architecture is structured in two primary layers:

- Data Layer: Comprises a collection of shards, with each shard storing and replicating records received from clients. Within each shard, each storage server acts as both a Primary for the

records it receives directly from clients and a Backup for records in the log segments of its peers in the same shard.

- Ordering Layer: Responsible for merging the locally ordered logs from each shard into a single global total order. To achieve this, the layer uses a hierarchical structure of aggregator nodes. Each leaf aggregator collects and merges records from a subset of storage servers. Higher-level aggregators then merge the outputs from lower levels, reducing the workload at any single node and enabling scalability to the system across multiple layers of the tree.

The global total order is determined using a mechanism called the **Global Cut**, a vector representing the durable record count for each shard. Periodically, storage servers issue information to the ordering layer regarding the number of records stored. The ordering layer computes the minimum count reported among all servers in each shard, forming the **Global Cut**, which is then broadcast to all storage servers. Each server applies a deterministic algorithm to assign sequence numbers to records included in the cut, ensuring all shards independently derive the same global order without additional coordination. This guarantees that every record has a unique global position even if written to different shards.

This persistence-first approach allows Scalog to seamlessly reconfigure without downtime or data loss. By decoupling data persistence from global ordering, Scalog avoids the bottlenecks associated with centralized sequencers and mitigates the downtime availability during reconfiguration. The hierarchical ordering layer further enhances scalability by distributing the computational load of ordering across multiple aggregator nodes. Clients are notified of their record's global sequence number once it is fully replicated and ordered, ensuring durability and consistency. However, while this architecture achieves high scalability and throughput, traversing the aggregator tree and its reliance on periodic cuts from the ordering layer introduces a large latency overhead.

3.2 Tree-Based Architectures

This section examines systems that, while most of them offer weaker guarantees than the previously discussed systems, introduce interesting architectural approaches to handling transactions. Unlike systems that enforce a strict global total order, these systems prioritize scalability and efficiency, frequently relaxing consistency guarantees in order to achieve better performance in geo-replicated environments. These systems achieve this by using trees in their design to efficiently manage data and transaction processing. By using hierarchical structures for data partitioning or for organizing transaction processing across distributed nodes, trees reduce the complexity of operations, improve scalability and enable efficient conflict resolution, specially in geo-replicated environment.

3.2.1 Eunomia

Eunomia [14] addresses the challenge of balancing throughput and visibility latency when implementing CC in geo-replicated systems. Similarly to Cure [15], Eunomia allows local updates to proceed independently, without requiring prior synchronization. Cure, a system designed to provide CC with high availability, relies on global stabilization procedures to track and enforce dependencies between updates. It uses VCs to precisely capture causal dependencies while optimizing visibility latency. Building on these ideas, Eunomia offers a more scalable approach by introducing a deferred local serialization procedure, which avoids intrusive synchronization overheads and reduces global stabilization costs.

Eunomia assumes that, in each data center, the object-space is divided into N partitions, each using a hybrid clock [16] to assign timestamps autonomously. Clients maintain a local variable tracking the largest observed timestamp during their session, representing their causal dependencies. When a new update is issued, the partition responsible assigns a timestamp as the maximum of three values: the current physical clock, the partition's last used timestamp incremented by one, and the client's timestamp incremented by one. This ensures that updates respect the causal order while avoiding synchronization overhead.

The log, central to Eunomia, tracks update IDs and their timestamps, enforcing CC by maintaining update order across partitions and regions. By separating metadata, update IDs and timestamps, from actual update content, Eunomia minimizes metadata transfer overhead between data centers. This design reduces inter-data center communication costs and allows partitions to operate independently while ensuring that all updates respect causal dependencies. Each data center maintains a VC, called *PartitionTime*, tracking the latest updates across partitions. Periodically, Eunomia computes a *StableTime*, the minimum value from *PartitionTime*, to identify updates that are stable - those whose causal dependencies into the log are fully satisfied. Stable updates are serialized into the log, ensuring that no updates with earlier timestamps can later arrive. This process allows Eunomia to achieve consistency while maintaining high throughput and avoiding the bottlenecks associated with global stabilization.

To support geo-replication, Eunomia incorporates a receiver module in each data center. The receiver is responsible for coordinating remote updates, ensuring that all causal dependencies are applied locally before forwarding updates to the relevant local partitions. The receiver maintains two different vectors: a queue of pending updates from each remote data center and a vector tracking the latest update operations locally applied from each remote data center. Updates are processed only when their dependencies and ordering constraints are resolved, ensuring consistency across geo-replicated data centers. Eunomia employs a red-black tree [17] to efficiently manage metadata update. This data structure enables fast insertion, deletion, and in-order traversal of updates, making it well-suited for handling large numbers of updates in high-throughput environments.

By removing synchronous global stabilization from the critical path, Eunomia achieves higher through-

put and lower visibility latency, particularly for local operations. However, the system still faces challenges in environments with clock skews, which can affect stabilization times.

3.2.2 Saturn

Saturn [5] is a metadata service designed for geo-replicated systems to enforce CC. It maintains a constant metadata size regardless of the number of clients, servers, partitions, or data centers involved. By decoupling metadata management from data dissemination, Saturn eliminates the trade-off between throughput and visibility latency and supports genuine Partial Replication (PR), enabling scalability as the number of geo-replication sites increases.

It introduces compact metadata units, called labels, which uniquely identify operations and ensure CC. Each client maintains a label that tracks their causal history, updated with every operation. This label is used for both updates and migrations between data centers, ensuring causal dependencies are maintained. Labels are compact and causally consistent, which simplifies metadata propagation and management across geo-replicated environments.

Saturn's architecture is composed by the following sub-components:

- Stateless Frontends: Act as intermediaries, intercepting client requests and forwarding them to the relevant storage servers.
- Gears: Attached to every storage server, they are in charge of intercepting operations, assigning corresponding labels with timestamps that respect causality. They propagate the associated data and metadata.
- Label Sink: A central point in each data center, it asynchronously collects labels from all gears, orders them based on timestamps, and forwards them for further processing.
- Remote Proxies: Implements updates originating from other data centers, thereby ensuring the integrity of the causal order.

To propagate metadata efficiently, Saturn uses a tree-based topology for metadata dissemination. This topology is managed by distributed serializers that minimize propagation latency. Data centers form the leaves of the tree, while serializers act as internal nodes, ensuring updates are visible with minimal delay. A key innovation is the *Weighted Minimal Mismatch* metric, which optimizes serializer placement and metadata paths to reduce delays while balancing performance-critical paths.

When building the tree, Saturn chooses the location of serializers from a predefined set of potential locations, such as existing data centers. Saturn employs a heuristic algorithm to build and refine the tree topology. Starting with a basic two-node tree, the algorithm incrementally adds new nodes, selecting configurations that minimize global mismatch. This adaptability allows the topology to scale and

adjust to changes in the number of data centers or shifts in workload characteristics. By limiting metadata propagation to only relevant branches of the tree, Saturn supports partial geo-replication, reducing metadata traffic and mitigating false dependencies.

Saturn achieves a balance between scalability and efficiency. Its tree-based topology reduces metadata latency while maintaining compact metadata size, and its PR capabilities optimize resource use.

3.2.3 ENGAGE

ENGAGE [6] is a storage system designed for partially replicated edge environments, aiming to support efficient session guarantees while addressing the challenges of low visibility times and bandwidth constraints. The system operates on a network of cloudlets, which are small-scale edge servers distributed geographically. These cloudlets replicate frequently accessed data to reduce latency for edge applications. Unlike traditional data centers, cloudlets are designed to operate closer to users, providing fast and localized data access while supporting PR to conserve resources.

ENGAGE combines VCs and a distributed metadata service to enforce CC and provide efficient session guarantees. It relies on a causally ordered log to ensure that operations dependent on each other are applied in the correct order, enabling low-latency updates. Each cloudlet tracks updates using two types of VCs: one capturing the causal history of each replicated object and another tracking the highest sequence numbers of updates received from remote cloudlets.

ENGAGE also introduces a hierarchical tree architecture to efficiently propagate metadata and update payloads. Leaf nodes in the tree correspond to the cloudlets, which serve client requests and synchronize updates. Inner nodes, deployed in data centers or larger cloudlets with more resources, aggregate metadata and optimize its dissemination, reducing communication delays and bandwidth usage. This tree architecture helps synchronize the log across all cloudlets, ensuring that updates are applied in the correct causal order and that dependencies between updates are respected.

Clients in ENGAGE interact with their preferred cloudlet, which processes their operations locally whenever possible. For read and write operations, clients specify session guarantees such as Read Your Writes or MRs. Additionally, clients also maintain two different VCs: one for updates the client has read and another for write operations they have performed. These clocks ensure that operations are executed consistently, even when clients migrate between cloudlets. Write operations involve assigning sequence numbers and updating metadata, while ensuring that all causal dependencies are resolved before updates are applied. In order to handle remote updates, each cloudlet maintains a list of pending updates. Updates are applied only when their dependencies have been satisfied, ensuring consistency across replicas without requiring excessive synchronization. Metadata dissemination is further optimized by combining update notifications with metadata flush messages, reducing signaling overhead by piggybacking VC updates onto other messages.

By combining VCs, session guarantees, and an efficient metadata service, ENGAGE achieves low visibility times and efficient bandwidth usage in edge environments.

3.2.4 A Distributed and Hierarchical Architecture for Deferred Validation of Transactions in Key-Value Stores

The proposed system [18] combines a hierarchical communication pattern, used in [19], with message batching techniques to build a transactional system capable of supporting more clients and achieve higher throughput. It seeks to mitigate bottlenecks caused by the need for total ordering transactions, assuming an optimistic concurrency control mechanism that performs best when transaction conflicts are rare.

Two concurrency control mechanisms are explored: one using timestamps, enforcing (1SR) by assigning commit timestamps, and another based on acquiring locks on the accessed objects during validation, aborting transactions in case of conflicts. Transactions begin locally on the client side without immediate communication with data nodes. On the one hand, read transactions operate on a global snapshot determined by the first read operation of the client, returning the latest object version consistent with the snapshot. On the other hand, write transactions are cached locally and only sent to data nodes during the commit phase. At this point, new writes are saved as tentative versions, validated either locally or hierarchically, in the case of distributed transactions, and subsequently committed or aborted, depending on the validation outcome.

The architecture is designed for distributed key-value storage systems, where data is partitioned across multiple data nodes. Validation nodes are organized hierarchically, with leaf nodes co-located with data nodes to handle local transactions. Higher-level nodes aggregate partial validation results for distributed transactions, forming a tree structure. This hierarchy ensures scalability by limiting the scope of communication and validation based on transaction locality, minimizing latency for transactions affecting only a small subset of nodes.

For local transactions, validation is performed at the leaf node co-located with the relevant data node, ensuring low-latency processing without requiring coordination with other nodes. Distributed transactions are validated hierarchically: leaf nodes perform partial validation and propagate results to higher-level nodes, which aggregate these results and perform full validation. This approach assumes high locality, where most transactions involve a small number of nodes, reducing the number of transactions requiring higher-level validation and improving throughput.

To further enhance performance, the system employs message batching, reducing the frequency of network communication and CPU overhead. By grouping multiple messages into a single transmission, batching minimizes the overhead associated with network protocols and improves overall efficiency. This distinguishes the proposed system from earlier architectures like [19], by enabling it to support

more clients and achieve higher throughput while maintaining low latency.

3.3 Locality in Distributed Logs

Although previous systems provide strong consistency, they can be expensive and inefficient in geo-replicated systems, since a total order often requires significant coordination between distant regions, leading to high latency.

This section discusses systems that do not impose a global total order. Rather, they use locality to partition data across regions, making them well-suited for geo-replicated environments. However, it also introduces trade-offs related to consistency and conflict resolution.

3.3.1 FuzzyLog

FuzzyLog [9] is a shared log abstraction that departs from traditional total ordering by employing a partial order of updates. This design choice enables scalable geo-distributed applications while offering flexible consistency guarantees, such as CC+. The system does not offer (Strong 1SR), instead focusing on (1SR) as a trade-off for improved scalability and availability.

At the core of its architecture is a Directed Acyclic Graph (DAG), where nodes represent updates and edges encode the happens-before relationships. The DAG is divided into colors, each corresponding to an independent shard of application data. Within each color, updates are organized into chains that are fully replicated across regions. Each region maintains its own local chain and lazily synchronizes with the chains of other regions, ensuring that operations can proceed independently with minimal cross-region coordination.

Clients interact with their local DAG replicas to either append updates or synchronize state. For single-color operations, a client appends a new update by specifying its content and the target color. This update is appended to the tail of the local chain for that color, with cross links to the last nodes observed by the client in other chains. These cross-links ensure causal dependencies are preserved, enabling serializable isolation across shards. Synchronization for a specific color involves fetching updates from other regions in reverse topological order, ensuring that the causal relationships encoded in the DAG are respected.

For multi-color transactions, FuzzyLog employs a coordination protocol that uses Skeen's algorithm [1]. This protocol ensures that updates spanning multiple shards are applied atomically and in a globally consistent order. The process involves two steps: first, each participating server proposes a timestamp for the transaction, based on its local logical clock and a unique nonce. The initiating client collects these proposals and selects the maximum timestamp as the global order for the transaction. In

the second step, the client disseminates this global timestamp back to the servers, which finalizes the transaction's placement in the DAG.

FuzzyLog delegates the responsibility for managing this coordination to clients, rather than a centralized sequencer, which aligns with its decentralized and scalable design. However, this approach introduces additional challenges, particularly for fault tolerance, as client failures can leave transactions in an incomplete state. To address this, FuzzyLog implements mechanisms such as Write-Ahead Logging (WAL) and recovery protocols, ensuring that transactions can be completed or recovered even in the presence of failures.

3.3.2 SLOG

SLOG [2] is a transactional key-value store designed to achieve (Strong 1SR), low-latency reads and writes, and high throughput for local transactions. In SLOG, a region consists of servers interconnected by a low-latency network, typically within a single data center. The system adopts a master-based architecture, where each data item has a designated "home" region, ensuring that writes and linearizable reads are directed to the master replica. Transactions are classified as either single-home when all accessed data resides in one region, eliminating the need for global coordination, or multi-home, when data spans multiple regions, requiring global coordination that increases latency.

To achieve (Strong 1SR), SLOG employs deterministic transaction execution. Transactions are pre-scheduled during a coordination phase, which establishes a global execution order outside transactional boundaries. This approach prevents runtime delays caused by cross-region conflicts and eliminates distributed deadlocks. However, this determinism imposes two specific requirements: the entire transaction must be present during the planning phase, where its execution order is determined, and its data access set must be known in advance.

Each region maintains a local log of transactions involving data it masters. These logs are exchanged between regions in batches, alongside sequence numbers to ensure that no batches are missed. This enables regions to reconstruct the local logs of others and guarantees a consistent ordering of transactions, interleaving them and creating a global log. Granules, which represent the smallest unit of data ownership in SLOG, can encompass either individual records or sorted ranges of records. To efficiently map granules to their home regions, SLOG employs a distributed index called Lookup Master, which also tracks how many times a granule's master region has changed. When a client submits a transaction, it is sent to the nearest region, regardless of where the data resides. The region consults the Lookup Master to identify the home regions for the accessed granules.

For single-home transactions, the transaction is forwarded to the corresponding home region, where it is appended to the local log to ensure consistency. Multi-home transactions, which span multiple regions, are sent to a designated region, that acts as a sequencer of the system, for coordination. This

designated region determines a global execution order for the transaction by consulting the Lookup Master to identify the home regions of the involved data granules. It then generates a special *LockOnlyTxn* transaction for each home region involved. These transactions acquire locks on the relevant data granules in the correct order and are appended to the respective local logs. This ensures consistent ordering across all regions. Transactions are executed in the order defined by each region's global log. While the global log is not totally ordered across regions, all executions are equivalent, strictly serializable ones.

The reliance on a central sequencer for multi-home transactions can increase latency when the sequencer is geographically distant from the other partitions. Additionally, this sequencer also creates a potential bottleneck if many multi-home transactions require global ordering simultaneously, especially when network latency between regions is high.

3.3.3 Detock

Detock [20] presents a graph-based concurrency control approach tailored for geo-partitioned databases, enabling independent and deterministic scheduling of single-home and multi-home transactions within each region. Unlike systems such as SLOG that rely on a global ordering mechanism, Detock embeds this coordination within the asynchronous exchange and replay of locally replicated logs. Each data item is assigned a home region which holds the primary copy of the data. Clients send transactions to their nearest region, which becomes the coordinator, responsible for determining the transaction's read and write sets and annotating them with home region details via a Home Directory, analogous to SLOG's Lookup Master.

Detock relies on dependency graphs for transaction execution. When a transaction is received by its home region, it is appended to a local log. Regions asynchronously exchange their local logs so that each region eventually receives the complete logs from all other regions, reconstructing a global log. From this, a dependency graph is constructed, allowing transactions to be executed in parallel while respecting sequential log order. This eliminates the need for global coordination, guaranteeing that transactions, regardless of type, require only a single round-trip between the initiating and participating regions, while also ensuring (Strong 1SR).

For single-home transactions, the dependency graph forms a DAG due to strict local ordering, allowing transactions to be executed once no incoming edges remain. For multi-home transactions, each region generates a special kind of transaction based on the parts of the transaction that involve its local data. These are appended to the local logs of the relevant regions, with their initial order determined independently by each region. This independent ordering may result in deadlocks, delaying their own execution and other conflicting single and multi-home transactions. Deadlocks caused by inconsistent ordering are resolved locally and deterministically, without aborting transactions or requiring cross-region communication. Although initial dependency graphs across regions may differ due to log interleaving,

they eventually converge to a consistent structure through deterministic conflict resolution.

Detock's performance suffers in skewed scenarios, where specific data items are accessed disproportionally, increasing the likelihood of deadlocks. While these are resolved deterministically, frequent conflicts can delay transaction processing. Additionally, asynchronous log replication and independent ordering may exacerbate contention in high-conflict workloads, further affecting performance.

3.4 Discussion

In this section we discuss previously presented systems. First, we will discuss these systems along three key dimensions: shared (totally ordered) logs, locality in distributed logs, and tree-based architectures. By doing this, we aim to evaluate the strengths and limitations of the systems, highlighting their trade-offs and implications for geo-distributed environments. Finally, we will provide an overall discussion to synthesize insights from these comparisons and identify areas for improvement.

3.4.1 Shared (Totally Ordered) Logs

Shared (totally ordered) logs provide strong consistency, making them an attractive abstraction for distributed systems.

Systems like CORFU and vCorfu rely on sequencers to ensure a total order of operations. While this approach is straightforward, it faces significant bottlenecks as the system size grows, with sequencers struggling to match increasing I/O bandwidth demands. Moreover, sequencers act as a single point of failure, limiting overall throughput and scalability. vCorfu enhances CORFU's design by adding transaction support, which increases its utility for applications requiring atomic operations. On the other hand, Scalog reduces the load on the centralized sequencer by aggregating information regarding updates via a hierarchical tree of aggregators, reducing bottlenecks and improving scalability. However, traversing the tree and aggregating updates introduces significant latency overheads.

More importantly, achieving a total order is inherently expensive in distributed systems. These solutions don't scale effectively, making them better suited for small clusters where the trade-offs between consistency, complexity, and scalability are less pronounced.

3.4.2 Tree-Based Architectures

Tree-based architectures present an effective alternative to address the challenges faced by systems like SLOG and Detock, especially in geo-distributed settings.

Systems such as Saturn, Eunomia and ENGAGE adopt tree-based designs to propagate metadata efficiently, enabling scalability while maintaining relaxed consistency guarantees. For example, Sat-

urn and Eunomia provide CC while ENGAGE supports session guarantees. However, none of these systems support transactions, limiting their applicability for workloads requiring strong consistency. In contrast, [18] employs a tree structure specifically for validating and committing transactions, enforcing (1SR). By handling single-partition transactions locally and validating multi-partition transactions hierarchically, this approach provides stricter guarantees than other tree-based systems.

Despite their different goals, all these systems share the common goal of optimizing transaction processing and metadata propagation in geo-replicated environments. By leveraging hierarchical designs, they effectively distributed validation and coordination, reducing bottlenecks often associated with centralized mechanisms.

3.4.3 Locality in Distributed Logs

Scaling shared logs in distributed systems often involves relaxing strict total order guarantees to reduce coordination overhead. For instance, SLOG and Detock enforce a global total ordering only on a subtype of transactions, whereas FuzzyLog focuses on (1SR) with more relaxed consistency guarantees.

FuzzyLog introduces the use of a DAG structure to maintain partial order, allowing single-home transactions to correspond to single-color updates and multi-home transactions to span multiple colors. Similarly, Detock uses a DAG for concurrency control but faces challenges such as deadlocks in skewed workloads due to its optimistic local ordering. SLOG, while leveraging a sequencer for global ordering, struggles in geo-replicated settings where high latency and scalability demands intensify bottlenecks caused by centralized coordination.

3.4.4 Overall Assessment

Table 3.1 outlines the key characteristics of the state-of-the-art systems previously discussed. We focus on three different aspects:

- **Consistency Model:** The consistency model is a critical aspect that determines the guarantees a system provides regarding the order and visibility of updates. Systems with strong consistency (e.g., CORFU, Scalog and SLOG) ensure (1SR) but often face significant coordination overhead in geo-distributed settings. In contrast, systems with weaker consistency models (e.g., Eunomia, Saturn and ENGAGE) prioritize scalability and efficiency but may limit applicability for workloads that require strict guarantees.

- **Transaction Support:** Transaction support is essential for systems that need to execute operations atomically across multiple partitions. Systems such as vCorfu, SLOG and Detock include robust transactional mechanisms, enabling them to handle operations that span multiple nodes.

Systems	Consistency Model	Transactions	Coordination
CORFU [3]	Strong	✗	Sequencer
Scalog [13]	Strong	✗	Sequencer w/Aggregators
vCorfu [4]	Strong	✓	Sequencer
Eunomia [14]	Weak	✗	Tree
Saturn [5]	Weak	✗	Tree
ENGAGE [6]	Weak	✗	Tree
[18]	Strong	✓	Tree
FuzzyLog [9]	Strong	✓	Skeen(Client-Based)
SLOG [2]	Strong	✓	Sequencer
Detock [20]	Strong	✓	Optimistic

Table 3.1: Comparison between systems presented in the Related Work

However, systems such as CORFU, Scalog and Saturn do not support transactions, which limits their applicability in use cases requiring atomicity.

- **Coordination Mechanism:** The coordination mechanism determines how a system maintains order and consistency. Sequencer-based systems, such as CORFU and vCorfu, rely on centralized components to enforce total order. While this simplifies the coordination process, it introduces significant bottlenecks in scalability due to the reliance on a single point of failure. Scalog mitigates this scalability bottleneck by aggregating information regarding updates, but incurs a large latency overhead in the process. Fuzzylog adopts a client-based mechanism using Skeen's algorithm, which eliminates centralized coordination but can introduce latency if the client is geographically distant from participating nodes. Optimistic approaches, like the one used in Detock, allow the system to locally order multi-home transactions in each region assuming convergence without immediate synchronization. While this reduces the coordination overhead, it introduces potential challenges in resolving conflicts, especially under skewed workloads. Finally, tree-based mechanisms leverage hierarchical designs to propagate metadata efficiently, reducing coordination overhead and enabling scalability.

Summary

This chapter surveyed systems that realize shared logs and geo-replicated transactions across different coordination styles—sequencer-based (e.g., CORFU, vCorfu, Scalog, SLOG), tree-based (e.g., Eunomia, Saturn, ENGAGE), optimistic (e.g., Detock), and genuine client/coordinator variants (e.g., FuzzyLog). The comparison highlights a central trade-off: centralized sequencers offer a simple ordering point but are highly sensitive to placement and load in wide-area settings, while decentralized and genuine

designs preserve locality at the cost of extra protocol steps and careful placement.

4

System Architecture

Contents

In this chapter, we present our architecture and the rationale behind its design. We explain how transactions are processed, separating single-home and multi-home paths, and introduce an algorithm to map the logical architecture to its physical deployment, optimizing coordinator placement across geo-locations.

With our proposed architecture, we aim to address limitations observed in prior designs, particularly those of sequencer-based and traditional tree-based solutions when supporting transactional execution in geo-distributed settings [2, 20]. Sequencer-based approaches impose a single choke point: all transactions must traverse one location, which is suboptimal when participants are geographically distant from the sequencer, inflating both communication costs and end-to-end latency. These systems can also struggle under skewed workloads, where certain regions experience disproportionate transaction volumes.

Tree-based architectures mitigate some of these issues by structuring coordination hierarchically, avoiding a single central point. However, they still face bottlenecks for multi-home transactions. A key observation motivating our design is that nodes positioned at the "edges" of the logical tree (far left/right) may be close in physical geography but far in the logical hierarchy, forcing transactions to traverse the entire tree and incurring unnecessary latency.

4.1 System Model and Assumptions

Our architecture targets a transactional key-value store partitioned across geo-replicated regions. Each region hosts several servers and can replicate both data and control information locally. Without loss of generality, we assume that replication and concurrency control within a region are managed by a Paxos-style algorithm [21] that maintains a local log that records transactions that affect its corresponding object. This approach guarantees that updates are consistently applied across all replicas within the location. The system supports PR: each data fragment may be replicated in a distinct subset of regions, and some data may reside in a single region only.

Each region is the "home" of a subset of objects, as in [2, 20]. Transactions that access data stored solely within one region are called *single-home* transactions; transactions that access data homed across multiple regions are called *multi-home* transactions.

Single-home transactions do not require cross-region coordination. They are ordered by the region's local shared log and executed according to the total order defined by Paxos.

Multi-home transactions are ordered in two phases. First, an inter-regional coordination protocol establishes a total order across all multi-home operations. After a multi-home transaction is ordered with respect to other multi-home transactions, it is inserted into the local logs of the participating regions, interleaving with local transactions at each region. Within each region, Paxos ensures that replicas interleave transactions in a mutually consistent way. As in systems such as Spanner [22], we assume the Paxos leader in each region also acts as that region's representative for inter-regional coordination. Each step of the global coordination algorithm is recorded in the local log so coordination can continue if the leader replica must be replaced.

4.2 Coordinator-Based Architecture for Total Order

To order multi-home transactions, we employ Skeen's algorithm [1], which assigns timestamps to distributed events and imposes a total order by taking the maximum across proposed timestamps. Concretely, each participating region computes a local timestamp and proposes it to a coordinator; the coordinator determines the transaction's global timestamp by computing the maximum between all pro-

posals, determining the transaction's position in the total order. Skeen's algorithm is *genuine* as defined by [7]: only the regions involved in a given transaction exchange messages to order it, which helps reduce latency.

A naive all-to-all execution of Skeen incurs quadratic message cost in the number of participants. Delegating timestamp aggregation to the client (as in [9]) removes that quadratic factor but may increase latency if the client is geographically distant from the participating regions, and it complicates fault tolerance. To address these challenges, we introduce a set of coordinators: dedicated entities that aggregate proposed timestamps for each relevant combination of regions. Regions communicate only with the appropriate coordinator for their transactions, keeping message complexity linear with the number of participants and avoiding reliance on distant clients.

In a system with n geo-locations, a coordinator is assigned to coordinate transactions involving all n locations. Additional coordinators manage subsets of locations, such as combinations of $n-1$ locations (with a total of $\binom{n}{n-1}$ such coordinators), combinations of $n-2$ locations (with $\binom{n}{n-2}$ coordinators), and so forth, down to combinations involving two geo-locations. This organization eliminates direct all-to-all communication among regions while ensuring that every multi-home transaction has a well-defined coordination endpoint.

Figure 4.1 provides a logical representation of the proposed architecture in a system composed of three distinct geo-locations. Each geo-location is responsible for managing a specific object ("A", "B" or "C"). The figure also illustrates the coordinators and their connections, highlighting how they interact. For example, a multi-home transaction involving geo-locations "A", "B" and "C" is assigned to the coordinator "ABC", which is responsible for determining a total order for transactions involving that combination of geo-locations.

In our design, Skeen's algorithm is executed collaboratively between regions and coordinators: regions propose timestamps; the designated coordinator for the involved combination of regions collects those proposals, computes the global timestamp as their maximum, and disseminates the result to the participants. Each region may then advance its logical clock if needed and enqueue the transaction for insert in the local shared log, ensuring that global transactions are consistently ordered across local logs.

At this stage, we assume fault tolerance will follow an approach similar to CORFU and NoPaxos [23]: upon a failure, the system briefly halts to reconfigure, ensuring that all participants agree on and are informed of the updated configuration (including coordinator assignments) before resuming.

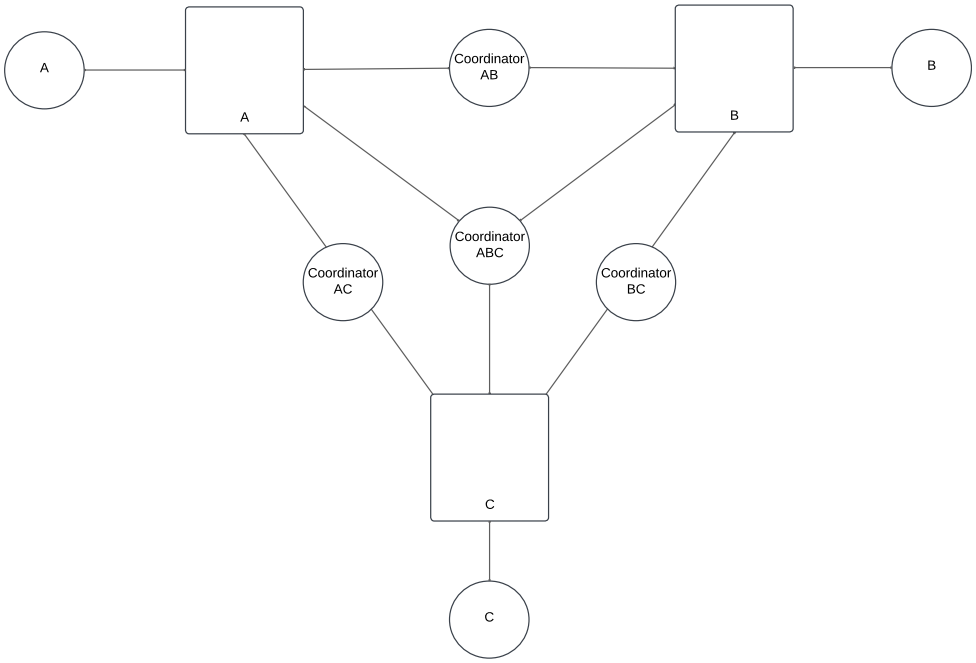

Figure 4.1: Logical Representation

4.3 Coordinator Placement

In some settings, the client submitting the transaction could act as Skeen's coordinator. We avoid this for two reasons: (i) if the client is remote, latency increases; (ii) client failures are harder to tolerate than server failures, whereas servers replicate their state via Paxos.

Coordinator selection can be random among the regions involved, but we evaluate an informed alternative that leverages topology (and, when available, load) to minimize latency. When a transaction is submitted, the system considers the involved regions and estimates the latency of Skeen's execution for each possible coordinator location, choosing the region that minimizes the estimated latency. This choice can be precomputed during system configuration. Thus, for every possible combination of regions, we assume a pre-assigned coordinator known to all participants.

Home regions are hosted in distinct data centers. Latencies are measured as round-trip times (RTT's) between data centers, so coordinators should be placed as centrally as possible with respect to the home regions they serve, minimizing the RTT of the message exchange for multi-home transactions.

Figure 4.2 illustrates this principle with three geo-locations, which in this example constitute the only data centers available to the system. Given the RTTs shown in the figure, the latency-minimizing choice for the three-way coordinator "ABC" is region "B", which is RTT-central with respect to "A" and "C". Placing the coordinator at "A" or "C" would systematically force one participant to traverse a longer RTT path, increasing end-to-end coordination time.

Because deployment is restricted to home-region data centers, the coordinator for a pair (e.g., "AB",

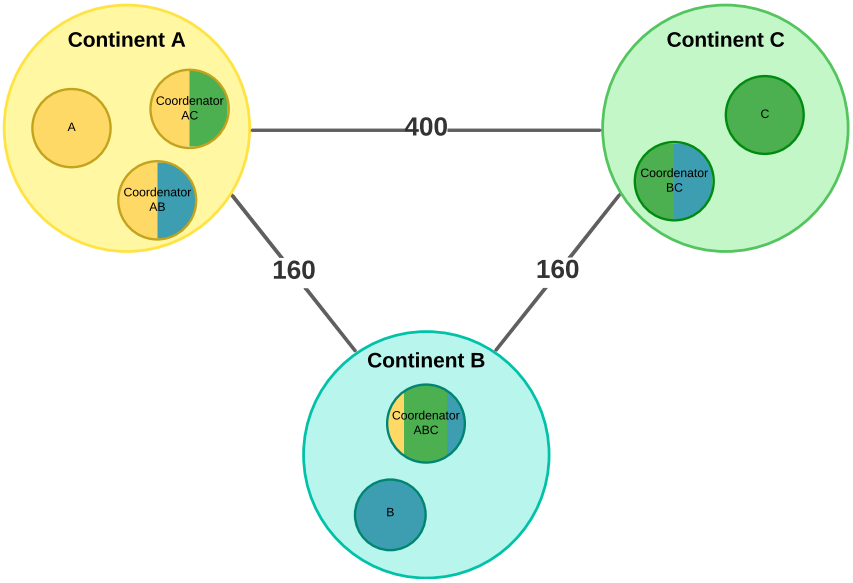

Figure 4.2: Selection of Coordinators (Physical Representation)

"BC", "AC") should be placed in one of the two regions involved ("AB" in "A" or "B"; "BC" in "B" or "C"; "AC" in "A" or "C"). If additional data centers were available, the preferred location would be the site that minimizes RTT to both regions—the latency "midpoint". Note that this midpoint could either be a third location or still coincide with one of the two regions when that region already minimizes the combined RTT. To avoid recreating a sequencer-like bottleneck, we also avoid concentrating all pairwise coordinators in a single geo-location.

In the current prototype, coordinator choices are configured statically. Making this process dynamic is not conceptually complex and is left for future work. In this version we use network latency as the sole criterion, but we plan to incorporate regional load to further balance the system.

4.4 Transaction Processing

Transactions are classified as single-home or multi-home, with distinct handling paths.

4.4.1 Single-Home Transactions

Single-home transactions involve only one region and can be processed and committed locally without communication with other regions or coordinators.

1. **Client Begins Transaction:** The client sends the transaction to the responsible region (e.g., a transaction over object "A" is sent to region "A").

2. **Local Check and Commit:** The region determines the transaction is single-home. Since no external coordination is needed, the transaction is inserted directly into the region's local log, which maintains a totally ordered sequence of transactions using a Paxos-based mechanism. This minimizes latency and enables fast, efficient processing of local operations.

4.4.2 Multi-Home Transactions

Multi-home transactions span multiple regions and require coordination to ensure a globally consistent total order and coherent interleaving with local transactions.

1. **Client Begins Transaction:** The client submits the transaction to all involved regions.

2. **Local Check and Forwarding to Coordinator:** Each region checks whether the transaction affects other regions. If so, it classifies the transaction as multi-home. Each region maintains a monotonically increasing logical clock that advances whenever it processes a new multi-home transaction. The region assigns the transaction a local timestamp strictly greater than any previously issued at that location. Regions also record, locally, the ordered set of transactions they have timestamped to ensure later that confirmations respect local order.

3. **Coordination via Skeen's Algorithm:** The involved regions and the pre-assigned coordinator for that combination (e.g., "AB" for "A" and "B") collaboratively run Skeen's algorithm. Each region sends its proposed timestamp to the coordinator, which aggregates proposals and computes the global timestamp as their maximum. This ensures a total order across participants while keeping communication overhead linear.

4. **Logical Clock Update:** Upon receiving the global timestamp, all participating regions advance their logical clocks as needed to avoid assigning future timestamps smaller than the agreed value.

5. **Order Check Before Commit:** Before committing a transaction, each region ensures there are no pending transactions to which it assigned smaller local timestamps. A transaction can only advance once all older-timestamped transactions have been confirmed, preserving local sequential consistency.

6. **Commit:** After the global timestamp is established, the transaction is added to a queue at each participating region. When it reaches the top, Paxos is used to insert the transaction into the local log in the globally agreed order.

Summary

This chapter presented a coordinator-based architecture for geo-distributed transactions. Each region maintains a Paxos-backed local log; multi-home transactions obtain a global order using Skeen's algorithm with pre-assigned coordinators for each combination of regions, and brief CORFU/NoPaxos-style reconfiguration provides fault tolerance. Coordinators are placed in RTT-central home-region data centers. Single-home transactions commit locally, while multi-home transactions proceed through proposal, aggregation, logical-clock update, and per-region commit, ensuring a globally consistent order that interleaves coherently with local work.

5

Evaluation

Contents

In this chapter, we evaluate the impact of carefully choosing the coordinator for each transaction under different levels of locality for global transactions. We compare our architecture against two baselines: SLOG [2], which uses a single centralized coordinator for all global transactions, and Skeen's algorithm with random coordinator selection.

To ensure a fair comparison, we reused and extended SLOG's public C++ codebase to build a shared foundation that supports both our approach and SLOG's centralized coordinator. Both variants run on the same networking substrate and common implementation, ensuring uniform evaluation conditions.

In SLOG, the ordering of global transactions is always performed by a single, preselected region, regardless of which regions the transaction touches. In our setup, that central coordinator is placed in Continent A, at *a0*. This fixed choice penalizes latency when the accessed data is far from *a0*.

Our workload consists exclusively of global transactions. We vary (i) the number of participating regions and (ii) the degree of locality among participants to study how locality shapes performance.

5.1 Experimental Setup

We use nine physical machines, each representing one region of the geo-distributed system. The same machines also host concurrent clients, with 9 client processes per machine. Each transaction reads and writes 9 objects, uniformly distributed across the regions it touches.

Contention is controlled via a dispersion parameter d, which governs the fraction of "hot" objects (i.e., those more frequently accessed). Larger d spreads accesses across more objects, reducing contention. Following the state-of-the-art [20], we set a high dispersion value of d=10000, corresponding to very low contention.

To emulate a realistic geo-distributed setting, we introduce artificial inter-region latencies. These values are hand-configured to reflect typical geographic relationships—lower RTTs within a continent and higher RTTs across continents. Figure 5.1 shows the system topology and configured RTTs between regions and continents. Region *a0* is chosen as SLOG's central coordinator (in red in Figure 5.1). In our system, the inter-region coordinator is placed at region *b0* (shown in yellow in Figure 5.1), which minimizes the RTT for these transactions under this topology. The central regions of each continent are *a0*, *b0*, and *c0*, respectively.

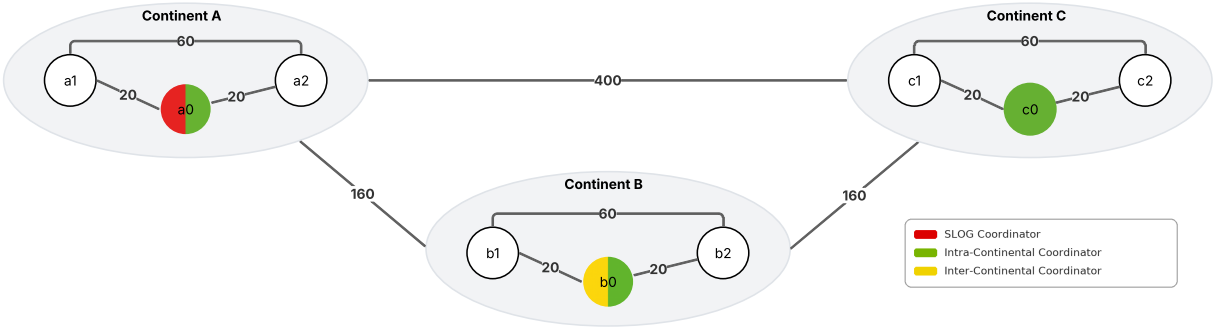

Figure 5.1: System topology. The red region denotes SLOG's central coordinator. Green regions denote coordinators for intra-continental Skeen transactions. The yellow region denotes the coordinator for inter-continental Skeen transactions.

5.2 Intra-Continental Locality

We begin with a high-locality scenario. In this experiment, global transactions interact only with regions within their origin continent, and they touch all regions in that continent. We refer to these as intra-continental transactions.

Figure 5.2 shows the CDF of end-to-end latency for global transactions originating from each continent: Continent A, Continent B, and Continent C. Under high locality, Skeen outperforms SLOG in regions far from the centralized sequencer, achieving up to $4\times$ latency improvement in Continent C. Although Skeen requires twice as many coordination steps as SLOG, the aggregate cost of intra-continental communication is significantly lower than inter-continental communication. This additional step cost is visible in Continent A, where SLOG performs better.

We also observe that the choice of coordinator within a continent significantly affects Skeen's performance. Consider Continent B, where *b0* is centrally located relative to *b1* and *b2*. Placing the coordinator at *b0* minimizes Skeen's execution time in two ways: (i) it reduces the protocol's worst-case round-trip (from start to the arrival of the final timestamp at all participants), directly lowering transaction latency from the client's perspective; and (ii) it shortens the time transactions remain tentative in regions, which helps unblock commits that are waiting behind concurrently ordered transactions. Together these effects reduce mean inter-region transaction latency by roughly 35%.

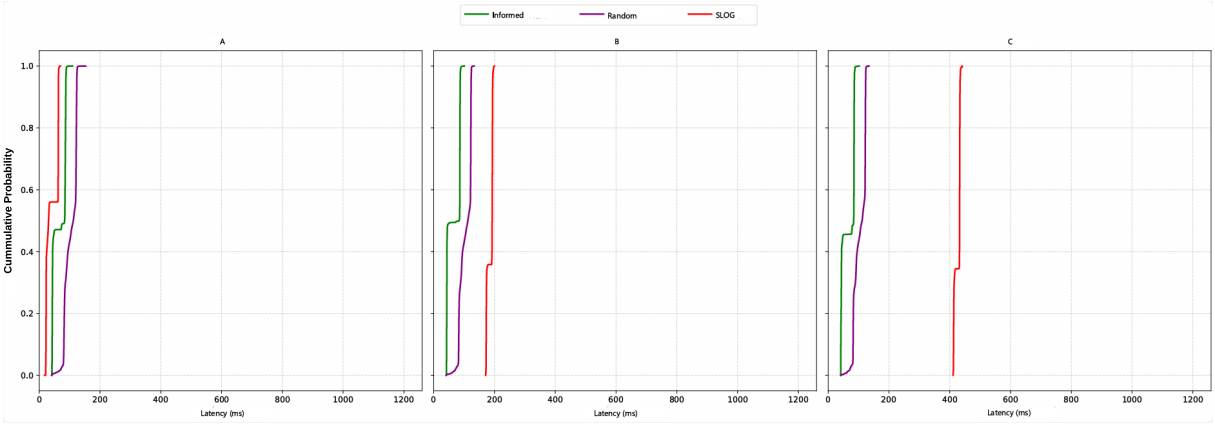

Figure 5.2: Per-region latency CDFs, comparing random coordinator selection with informed selection of the most central region.

5.3 Inter-Continental Locality

We now consider a mixed scenario in which, besides intra-continental transactions, a small fraction (10%) of transactions are inter-continental, touching one region from each continent. The choice of 10% reflects typical edge workloads in which the vast majority of transactions are intra-region or locality-preserving (e.g. 90%) [24, 25]; we introduce a modest inter-continental tail to stress the system under realistic asymmetry.

Figure 5.3 reports the latency CDF per region. Introducing inter-continental transactions significantly affects Skeen-based protocols, while SLOG remains comparatively stable. The degradation arises from

the large disparity between intra- and inter-continental communication times. Intra-continental transactions obtain their final timestamps much faster than inter-continental ones, creating queues that wait for the final global order of slow inter-continental operations—i.e., the convoy effect.

In the presence of this effect, the latency gains of Skeen diminish substantially: Skeen exhibits worse mean latency than SLOG in Continent A and Continent B. However, both systems still show better mean latency in Continent C, where the communication cost to the Continent A–based centralized coordinator dominates SLOG's end-to-end time.

While informed coordinator selection still helps, its impact is reduced by the convoy effect, yielding only about a 10% mean-latency improvement for transactions originating in Continent A and Continent C. The gains remain significant in Continent B—about 40% versus random—because the informed choice more strongly shortens Skeen's execution time there.

For a Continent B–origin inter-continental transaction, the time the transaction remains tentative in Continent B is approximately 360, 160, and 360 milliseconds when the Skeen coordinator is in Continent A, Continent B, or Continent C, respectively—an average of 293 ms under random choice, about 80% higher than the constant 160 ms under informed placement. In contrast, for Continent C– and Continent A–origin transactions, the average benefit of informed selection over random is only about 7%, as coordination time is dominated by the Continent C and Continent A communication cost regardless of where the coordinator is placed.

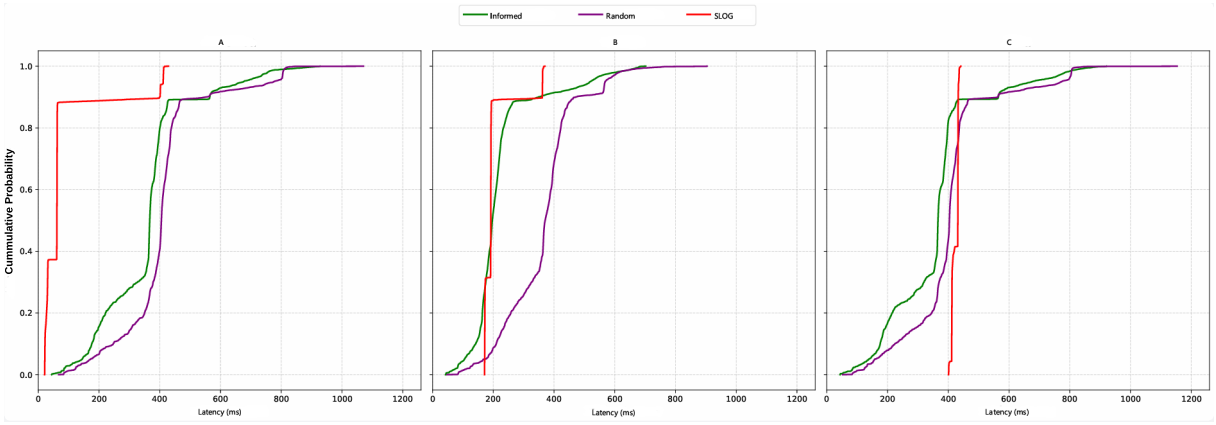

Figure 5.3: Latency CDFs by region, comparing the impact of coordinator selection in the presence of inter-region transactions.

5.4 Visualizing the Convoy Effect

To make the convoy effect concrete, Figure 5.4 shows a timeline at region *c1*. A slower inter-continental transaction $T0$ completes after long RTTs, while a faster intra-continental transaction $T1$ arrives later but cannot be confirmed because $T0$ holds a smaller local timestamp. The shaded interval highlights the

waiting time before $T1$ can be confirmed, which accumulates across many such cases and shifts the CDFs reported above.

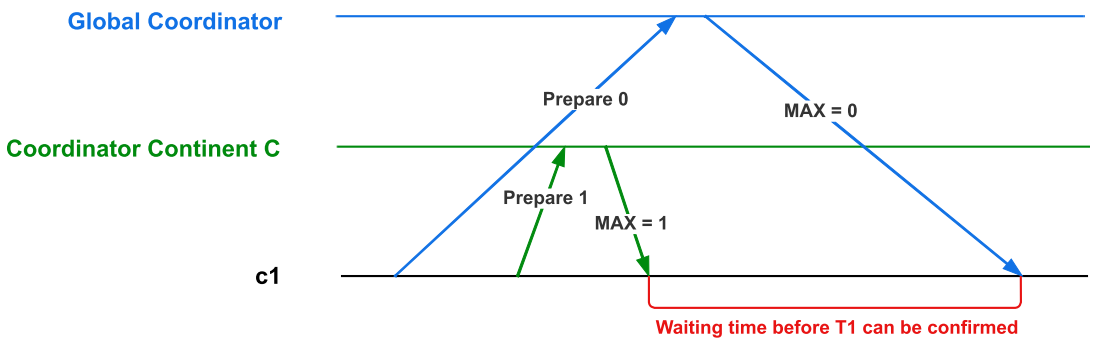

Figure 5.4: Illustrative timeline at region *c1*. The shaded interval marks the waiting time before $T1$ can be confirmed. A slower inter-continental $T0$ finalizes after long RTTs; a faster intra-continental $T1$ must wait because $T0$ has the smaller local timestamp.

5.5 Isolating the Convoy Effect

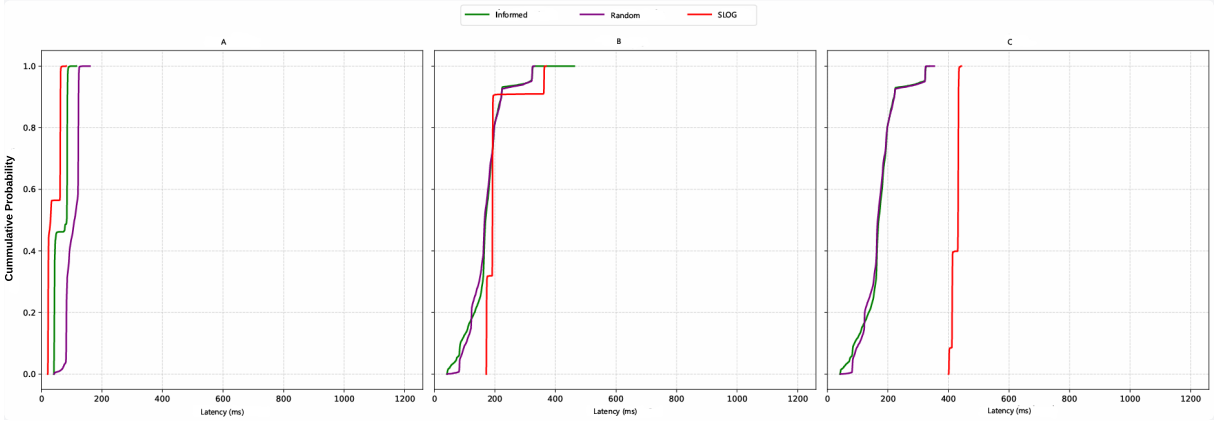

Figure 5.5: Per-region latency CDFs with inter-continental transactions limited to Continent B–Continent C. Removing the Continent C–A leg reduces Skeen's execution time and its convoy.

To isolate the effect, we modify the mixed workload so inter-continental transactions span only Continent B and Continent C (one region from each), excluding Continent A. Because these transactions touch two continents, the coordinator for such transactions is chosen at random between the two involved regions.

Figure 5.5 shows per-region CDFs. Skeen-based systems perform markedly better in Continent B and Continent C than in the previous mixed case. The improvement stems from removing the slow Continent C–A leg from inter-continental coordination, which shortens Skeen's execution time and mitigates

its associated convoy. In continents still affected by the convoy effect, informed selection offers little advantage over random—the slow path dictates the critical path, overshadowing intra-continental gains.

Summary

In this chapter, we experimentally evaluated our coordinator-based architecture under varying transaction locality. Using a public C++ codebase, we compared it to SLOG. We found that informed coordinator placement allows Skeen's algorithm to substantially outperform SLOG. Under mixed workloads with a realistic 10% share of inter-continental transactions (consistent with prior reports of 90% locality-preserving traffic), RTT asymmetry combined with local admission rules induces a convoy effect that reduces these gains. Overall, locality-aware coordinator selection improves latency and robustness to asymmetry, but even a modest fraction of long inter-continental transactions can make the convoy effect dominant.

6

Conclusions and Future Work

This dissertation investigated genuine total ordering for geo-distributed transactional systems, focusing on Skeen's algorithm and the role of informed coordinator selection. A coordinator-based design was presented to retain linear message complexity, together with a physical placement principle grounded in RTT-centrality among participating home regions. Evaluation showed substantial latency gains under high intra-continental locality while also exposing a limiting factor: in mixed workloads, RTT asymmetry combined with local admission rules induces a convoy effect that diminishes Skeen's advantages and can favor a centralized sequencer in some regions. Overall, informed coordinator placement is effective but insufficient on its own; robustness in heterogeneous wide-area settings requires additional mechanisms that explicitly address convoy formation.

Future research should address two complementary directions. First, the system can be extended to support an online placement algorithm that automatically and dynamically optimizes end-to-end RTT for each region combination, taking into account the network delays and workload characterization. Second, to include mechanisms to mitigate the convoy effect, in particular on transactions that involve a single region. These directions aim to turn informed coordination and convoy mitigation into robust building blocks for geo-replicated transactional systems, maintaining strict correctness while improving performance across diverse, asymmetric network conditions.

Bibliography

[1] K. P. Birman and T. A. Joseph, “Reliable communication in the presence of failures,” *ACM Trans. Comput. Syst.*, vol. 5, no. 1, pp. 47–76, 1987.

[2] K. Ren, D. Li, and D. J. Abadi, “SLOG: serializable, low-latency, geo-replicated transactions,” *Proceedings of the VLDB Endowment*, vol. 12, no. 11, pp. 1747–1761, 2019.

[3] M. Balakrishnan, D. Malkhi, J. D. Davis, V. Prabhakaran, M. Wei, and T. Wobber, “CORFU: A distributed shared log,” *ACM Trans. Comput. Syst.*, vol. 31, no. 4, p. 10, 2013.

[4] M. Wei, A. Tai, C. J. Rossbach, I. Abraham, M. Munshed, M. Dhawan, J. Stabile, U. Wieder, S. Fritchie, S. Swanson, M. J. Freedman, and D. Malkhi, “vcorfu: A cloud-scale object store on a shared log,” in *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, Mar. 2017.

[5] M. Bravo, L. E. T. Rodrigues, and P. V. Roy, “Saturn: A distributed metadata service for causal consistency,” in *Proceedings of the Twelfth European Conference on Computer Systems (EuroSys)*, Apr. 2017.

[6] M. Belém, P. Fouto, T. Lykhenko, J. Leitão, N. M. Preguiça, and L. Rodrigues, “Engage: Session guarantees for the edge,” in *31st International Conference on Computer Communications and Networks, (ICCCN)*, Jul. 2022.

[7] R. Guerraoui and A. Schiper, “Genuine atomic multicast in asynchronous distributed systems,” *Theoretical Computer Science*, vol. 254, no. 1, pp. 297–316, 2001.

[8] L. Lamport, “Time, clocks, and the ordering of events in a distributed system,” *Commun. ACM*, vol. 21, no. 7, p. 558–565, 1978.

[9] J. Lockerman, J. Faleiro, J. Kim, S. Sankaran, D. Abadi, J. Aspnes, S. Sen, and M. Balakrishnan, “The FuzzyLog: A partially ordered shared log,” in *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, Oct. 2018, pp. 357–372.

[10] S. Gilbert and N. A. Lynch, “Brewer’s conjecture and the feasibility of consistent, available, partition-tolerant web services,” *SIGACT News*, vol. 33, no. 2, pp. 51–59, 2002.

[11] D. B. Terry, A. J. Demers, K. Petersen, M. Spreitzer, M. Theimer, and B. B. Welch, “Session guarantees for weakly consistent replicated data,” in *Proceedings of the Third International Conference on Parallel and Distributed Information Systems*, Sep. 1994.

[12] P. Viotti and M. Vukolic, “Consistency in non-transactional distributed storage systems,” *ACM Comput. Surv.*, vol. 49, no. 1, pp. 19:1–19:34, 2016.

[13] C. Ding, D. Chu, E. Zhao, X. Li, L. Alvisi, and R. van Renesse, “Scalog: Seamless reconfiguration and total order in a scalable shared log,” in *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*, Feb. 2020.

[14] C. Gunawardhana, M. Bravo, and L. E. T. Rodrigues, “Unobtrusive deferred update stabilization for efficient geo-replication,” in *2017 USENIX Annual Technical Conference (ATC)*, Jul. 2017.

[15] D. D. Akkoorath, A. Z. Tomsic, M. Bravo, Z. Li, T. Crain, A. Bieniusa, N. M. Preguiça, and M. Shapiro, “Cure: Strong semantics meets high availability and low latency,” in *2016 IEEE 36th International Conference on Distributed Computing Systems (ICDCS)*, Jun. 2016.

[16] S. S. Kulkarni, M. Demirbas, D. Madappa, B. Avva, and M. Leone, “Logical physical clocks,” in *Principles of Distributed Systems: 18th International Conference, OPODIS 2014*, Dec. 2014.

[17] L. J. Guibas and R. Sedgewick, “A dichromatic framework for balanced trees,” in *19th Annual Symposium on Foundations of Computer Science*. IEEE, 1978, pp. 8–21.

[18] J. Amaro, “A distributed and hierarchical architecture for deferred validation of transactions in key-value stores,” 2018.

[19] J. Grov and P. C. Ölveczky, “Scalable and fully consistent transactions in the cloud through hierarchical validation,” in *International Conference on Data Management in Cloud, Grid and P2P Systems*, Aug. 2013.

[20] C. D. T. Nguyen, J. K. Miller, and D. J. Abadi, “Detock: High performance multi-region transactions at scale,” *Proceedings of the ACM on Management of Data*, vol. 1, no. 2, pp. 1–27, 2023.

[21] L. Lamport, “Paxos made simple, fast, and byzantine,” in *Procedings of the 6th International Conference on Principles of Distributed Systems.*, 2002, pp. 7–9.

[22] J. C. Corbett, J. Dean, M. Epstein, A. Fikes, C. Frost, J. J. Furman, S. Ghemawat, A. Gubarev, C. Heiser, P. Hochschild, W. C. Hsieh, S. Kanthak, E. Kogan, H. Li, A. Lloyd, S. Melnik, D. Mwaura,

D. Nagle, S. Quinlan, R. Rao, L. Rolig, Y. Saito, M. Szymaniak, C. Taylor, R. Wang, and D. Woodford, "Spanner: Google's globally distributed database," *ACM Trans. Comput. Syst.*, vol. 31, no. 3, p. 8, 2013.

[23] J. Li, E. Michael, N. K. Sharma, A. Szekeres, and D. R. K. Ports, "Just say NO to paxos overhead: Replacing consensus with network ordering," in *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, Nov. 2016.

[24] A. Ailijiang, A. Charapko, M. Demirbas, and T. Kosar, "Wpaxos: Wide area network flexible consensus," *IEEE Trans. Parallel Distributed Syst.*, vol. 31, no. 1, pp. 211–223, 2020.

[25] Y. Lin, B. Kemme, M. Patiño-Martínez, and R. Jiménez-Peris, "Enhancing edge computing with database replication," in *26th IEEE Symposium on Reliable Distributed Systems (SRDS 2007), Beijing, China, October 10-12, 2007*, 2007, pp. 45–54.