WISCONSIN
UNIVERSITY OF WISCONSIN–MADISON

CS 764: Topics in Database Management Systems
Lecture 21: Replication

Xiangyao Yu

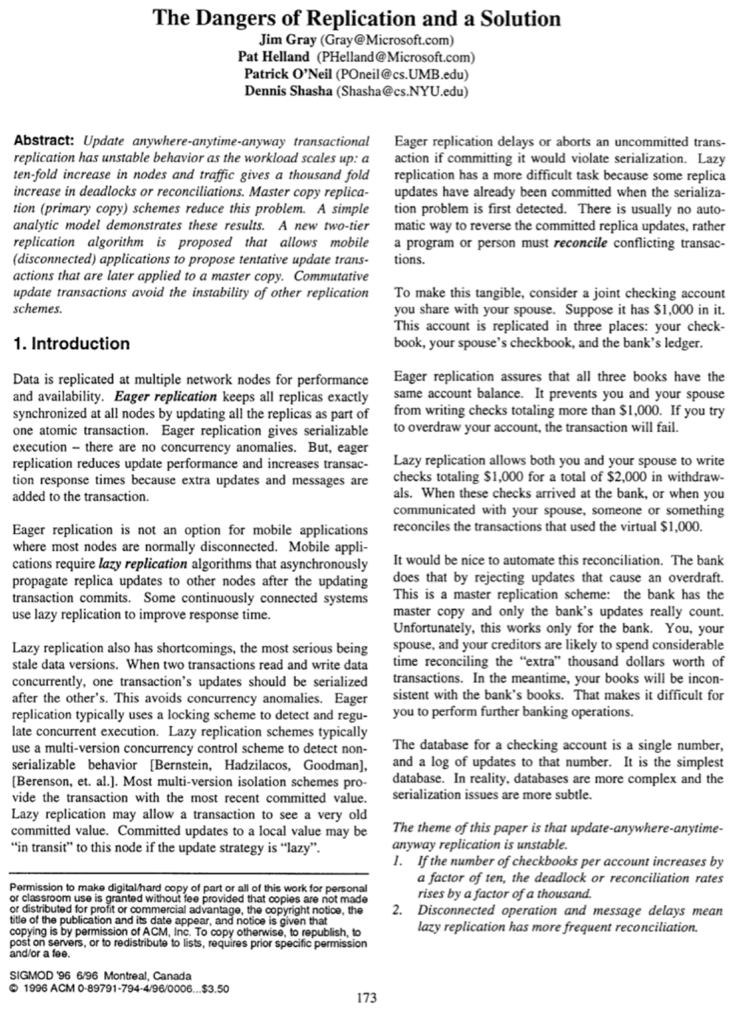
11/17/2021

Announcement

Each group meets with the instructor for 10 min to discuss the final project

- Pick meeting time using the following google sheet: https://docs.google.com/spreadsheets/d/1Hs-1rXegGThZTyD0IytMi4SkPVVqJl-T0e5PP7bOeLg/edit?usp=sharing
- Please email the instructor if need a different time

Today's Paper: Replication



The Dangers of Replication and a Solution

Jim Gray (Gray@Microsoft.com)
Pat Helland (PHelland@Microsoft.com)
Patrick O'Neil (POneil@cs.UMB.edu)
Dennis Shasha (Shasha@cs.NYU.edu)

Abstract: *Update anywhere-anytime-anyway transactional replication has unstable behavior as the workload scales up: a ten-fold increase in nodes and traffic gives a thousand fold increase in deadlocks or reconciliations. Master copy replication (primary copy) schemes reduce this problem. A simple analytic model demonstrates these results. A new two-tier replication algorithm is proposed that allows mobile (disconnected) applications to propose tentative update transactions that are later applied to a master copy. Commutative update transactions avoid the instability of other replication schemes.*

1. Introduction

Data is replicated at multiple network nodes for performance and availability. ***Eager replication*** keeps all replicas exactly synchronized at all nodes by updating all the replicas as part of one atomic transaction. Eager replication gives serializable execution – there are no concurrency anomalies. But, eager replication reduces update performance and increases transaction response times because extra updates and messages are added to the transaction.

Eager replication is not an option for mobile applications where most nodes are normally disconnected. Mobile applications require ***lazy replication*** algorithms that asynchronously propagate replica updates to other nodes after the updating transaction commits. Some continuously connected systems use lazy replication to improve response time.

Lazy replication also has shortcomings, the most serious being stale data versions. When two transactions read and write data concurrently, one transaction's updates should be serialized after the other's. This avoids concurrency anomalies. Eager replication typically uses a locking scheme to detect and regulate concurrent execution. Lazy replication schemes typically use a multi-version concurrency control scheme to detect non-serializable behavior [Bernstein, Hadzilacos, Goodman], [Berenson, et. al.]. Most multi-version isolation schemes provide the transaction with the most recent committed value. Lazy replication may allow a transaction to see a very old committed value. Committed updates to a local value may be "in transit" to this node if the update strategy is "lazy".

SIGMOD '96 6/96 Montreal, Canada
© 1996 ACM 0-89791-794-4/96/0006...$3.50

Eager replication delays or aborts an uncommitted transaction if committing it would violate serialization. Lazy replication has a more difficult task because some replica updates have already been committed when the serialization problem is first detected. There is usually no automatic way to reverse the committed replica updates, rather a program or person must ***reconcile*** conflicting transactions.

To make this tangible, consider a joint checking account you share with your spouse. Suppose it has $1,000 in it. This account is replicated in three places: your checkbook, your spouse's checkbook, and the bank's ledger.

Eager replication assures that all three books have the same account balance. It prevents you and your spouse from writing checks totaling more than $1,000. If you try to overdraw your account, the transaction will fail.

Lazy replication allows both you and your spouse to write checks totaling $1,000 for a total of $2,000 in withdrawals. When these checks arrived at the bank, or when you communicated with your spouse, someone or something reconciles the transactions that used the virtual $1,000.

It would be nice to automate this reconciliation. The bank does that by rejecting updates that cause an overdraft. This is a master replication scheme: the bank has the master copy and only the bank's updates really count. Unfortunately, this works only for the bank. You, your spouse, and your creditors are likely to spend considerable time reconciling the "extra" thousand dollars worth of transactions. In the meantime, your books will be inconsistent with the bank's books. That makes it difficult for you to perform further banking operations.

The database for a checking account is a single number, and a log of updates to that number. It is the simplest database. In reality, databases are more complex and the serialization issues are more subtle.

The theme of this paper is that update-anywhere-anytime-anyway replication is unstable.

1. *If the number of checkbooks per account increases by a factor of ten, the deadlock or reconciliation rates rises by a factor of a thousand.*
2. *Disconnected operation and message delays mean lazy replication has more frequent reconciliation.*

SIGMOD 1996

Data Replication

Goal 1: High availability (HA)

- When a server fails, another replica can serve the requests (users do not notice the failure)
- High availability vs. durability

Data Replication

Goal 1: High availability (HA)

- When a server fails, another replica can serve the requests (users do not notice the failure)
- High availability vs. durability

Goal 2: Performance

- All replicas can serve (sometimes readonly) requests
- Can choose the geographically closest replica

Replication Methods — Lazy vs. Eager

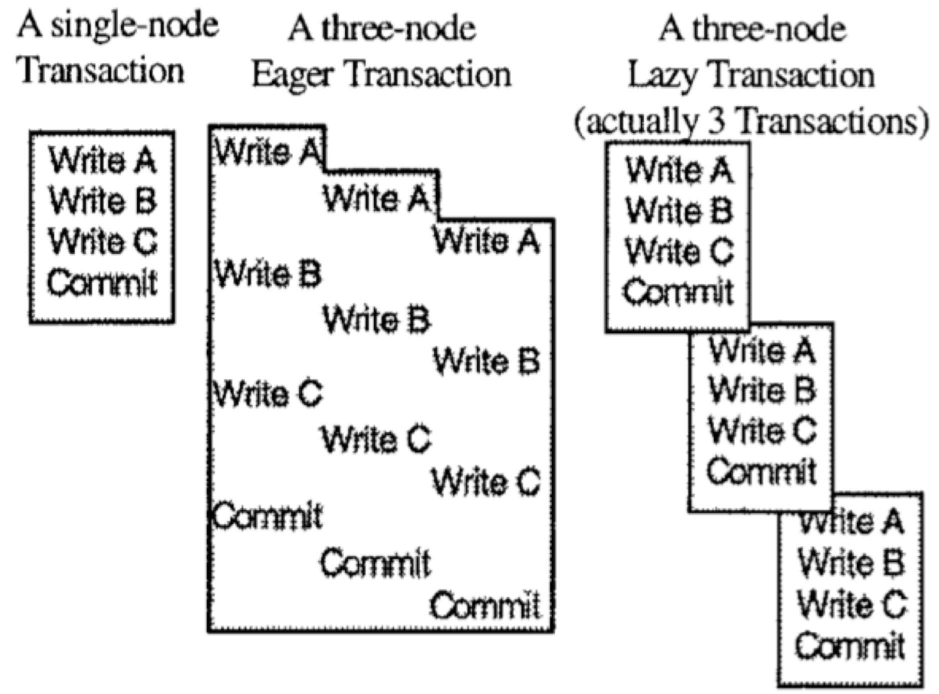
Eager

- Transaction updates records in all replicas

Lazy

- Transaction runs on one node and commits.
- Replication happens in the background as separate transactions.

Figure 1: When replicated, a simple single-node transaction may apply its updates remotely either as part of the same transaction (*eager*) or as separate transactions (*lazy*). In either case, if data is replicated at *N* nodes, the transaction does *N* times as much work



Replication Methods — Group vs. Master

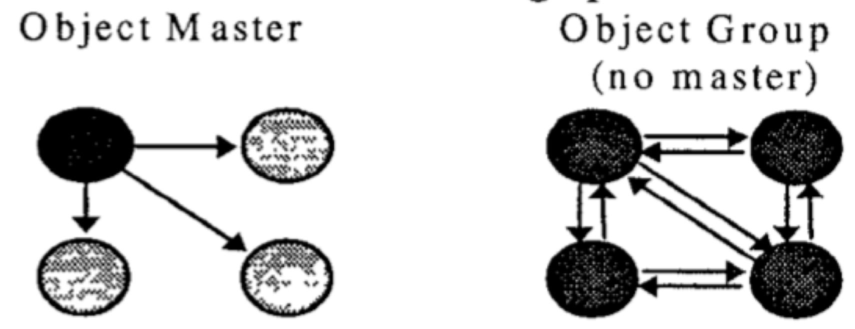
Master (**Single-Master**)

- Each record has a single master node.
- The record can be updated only in the master node.

Group (**Multi-Master**)

- Each record can be updated in multiple nodes.

Figure 2: Updates may be controlled in two ways. Either all updates emanate from a master copy of the object, or updates may emanate from any. Group ownership has many more chances for conflicting updates.



Replication Methods

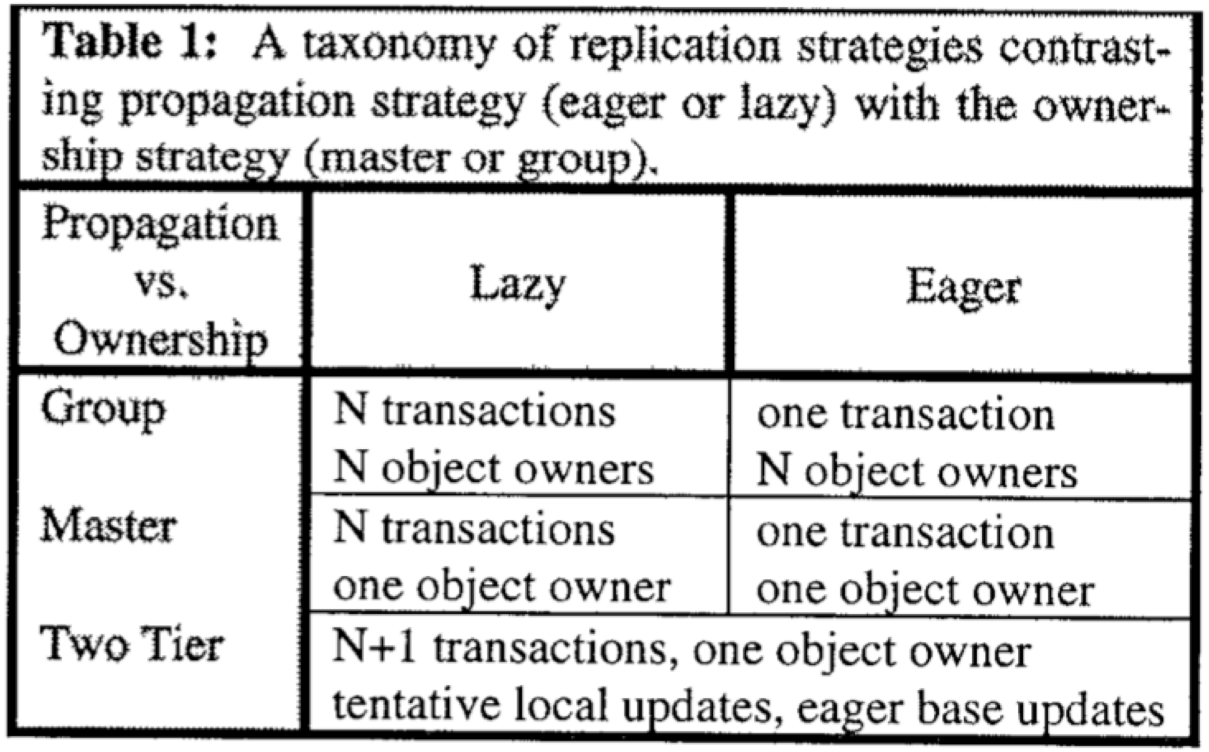
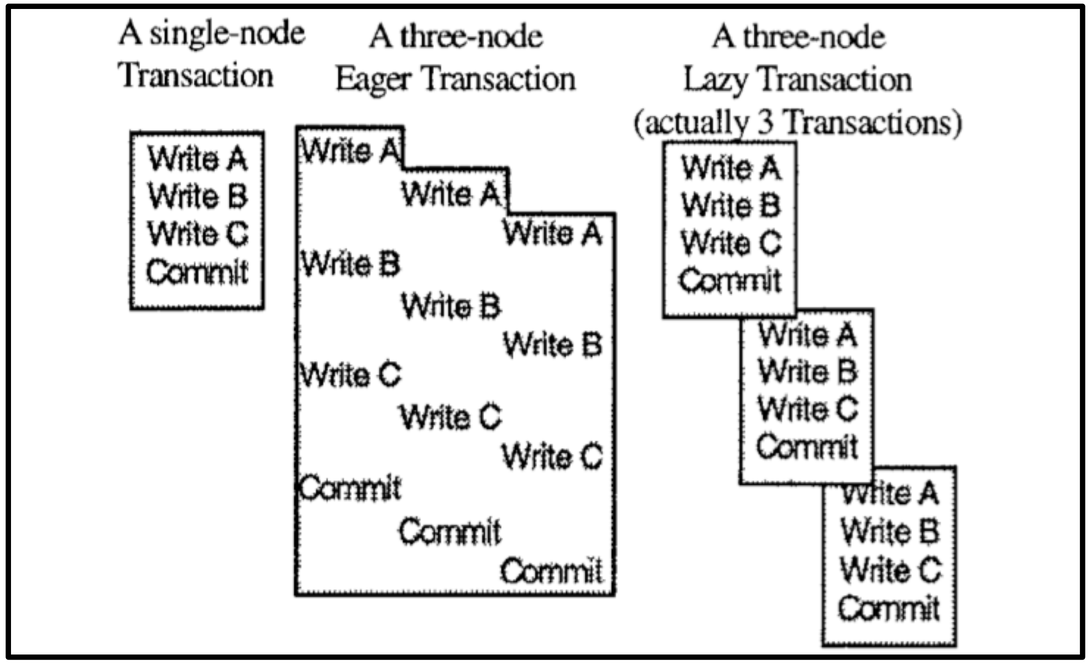
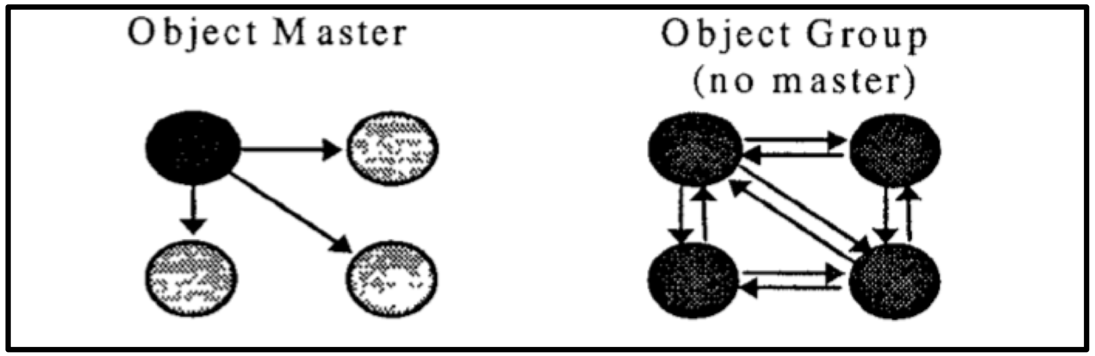


Table 1: A taxonomy of replication strategies contrasting propagation strategy (eager or lazy) with the ownership strategy (master or group).

Propagation vs. Ownership	Lazy	Eager
Group	N transactions N object owners	one transaction N object owners
Master	N transactions one object owner	one transaction one object owner
Two Tier	N+1 transactions, one object owner tentative local updates, eager base updates	





How do modern systems handle replication?

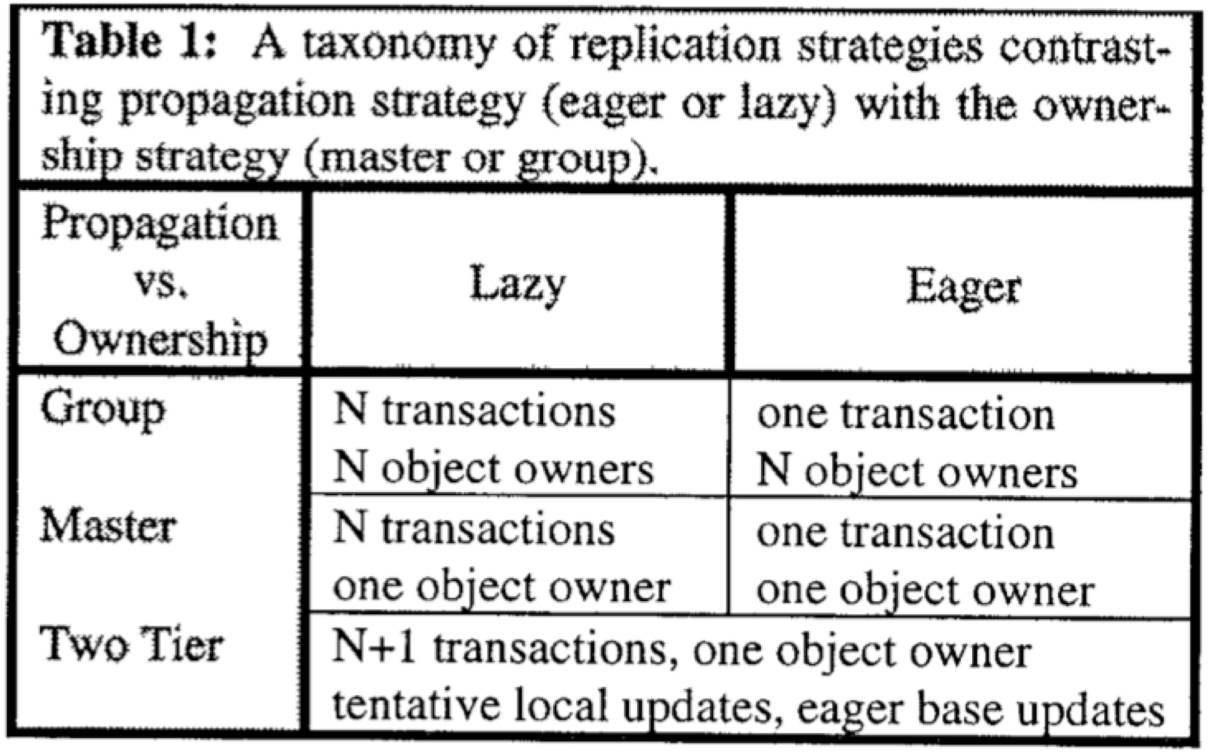


Table 1: A taxonomy of replication strategies contrasting propagation strategy (eager or lazy) with the ownership strategy (master or group).

Propagation vs. Ownership	Lazy	Eager
Group	N transactions N object owners	one transaction N object owners
Master	N transactions one object owner	one transaction one object owner
Two Tier	N+1 transactions, one object owner tentative local updates, eager base updates	

Single-Master Replication

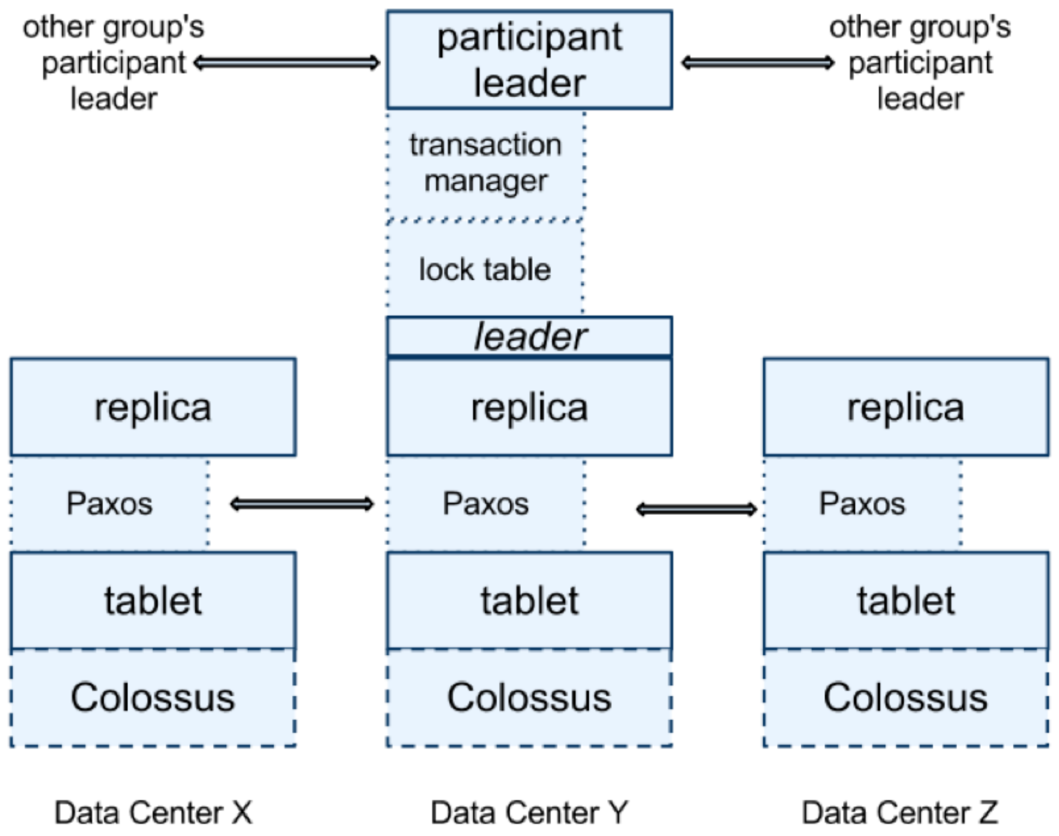
Modern single-master solutions

- Active-Passive
- Active-Active

Single-Master Replication — Active-Passive

Google Spanner

- Transaction logic is processed in the leader replica
- Updates replicated to other replicas through Paxos
- **No locking performed in backup replicas**

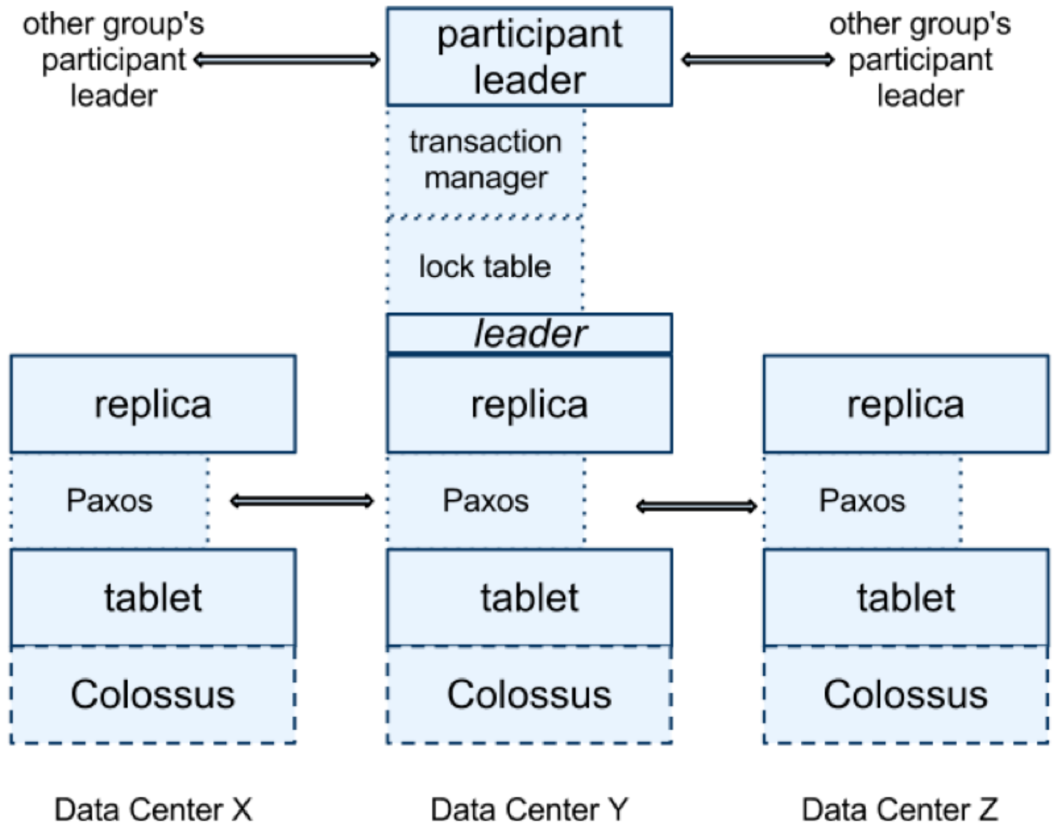


Source: *Spanner: Google's Globally-Distributed Database*, OSDI 2012

Single-Master Replication — Active-Passive

Google Spanner

- Transaction logic is processed in the leader replica
- Updates replicated to other replicas through Paxos
- **No locking performed in backup replicas**
- All replicas can serve readonly transactions
- Only leaders serve read-write transactions
- Transaction commits after replication entirely done

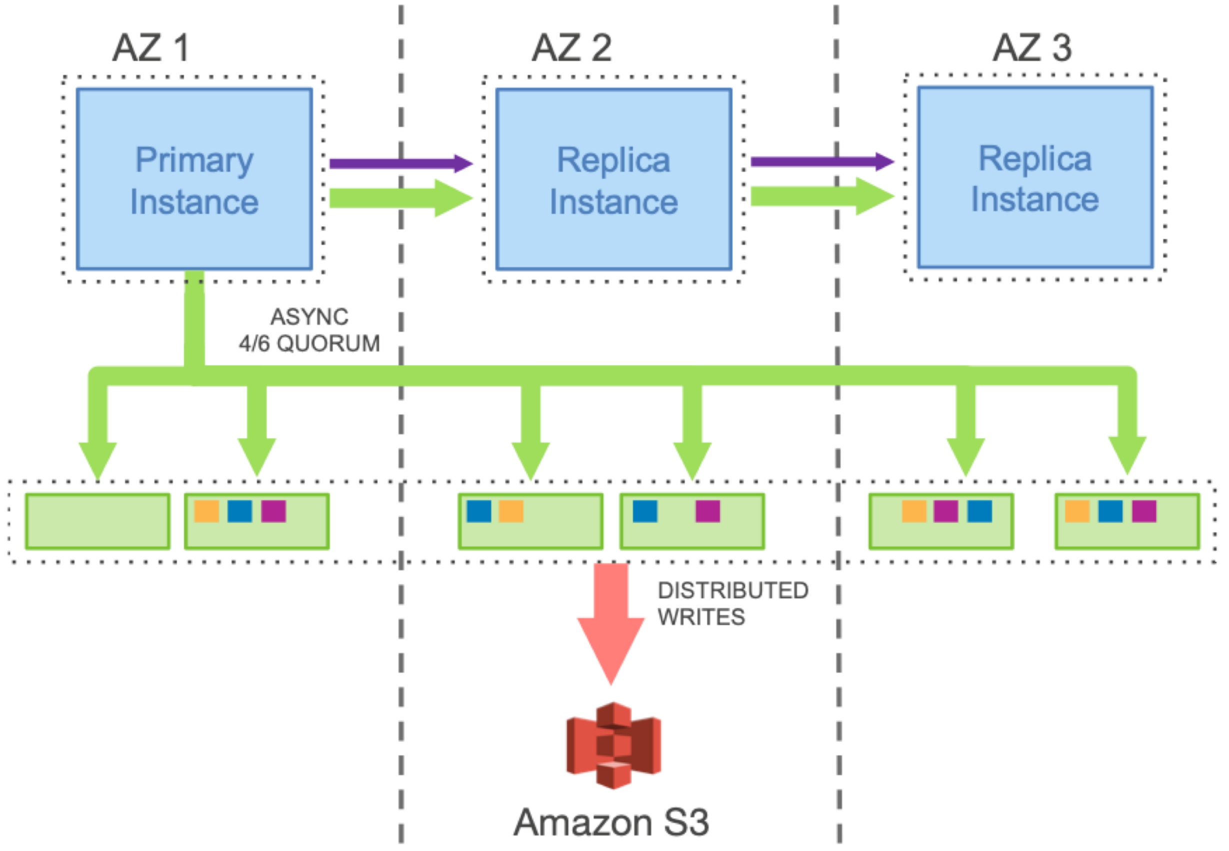


Source: *Spanner: Google's Globally-Distributed Database*, OSDI 2012

Single-Master Replication — Active-Passive

Amazon Aurora

- Single primary instance, multiple replica instances

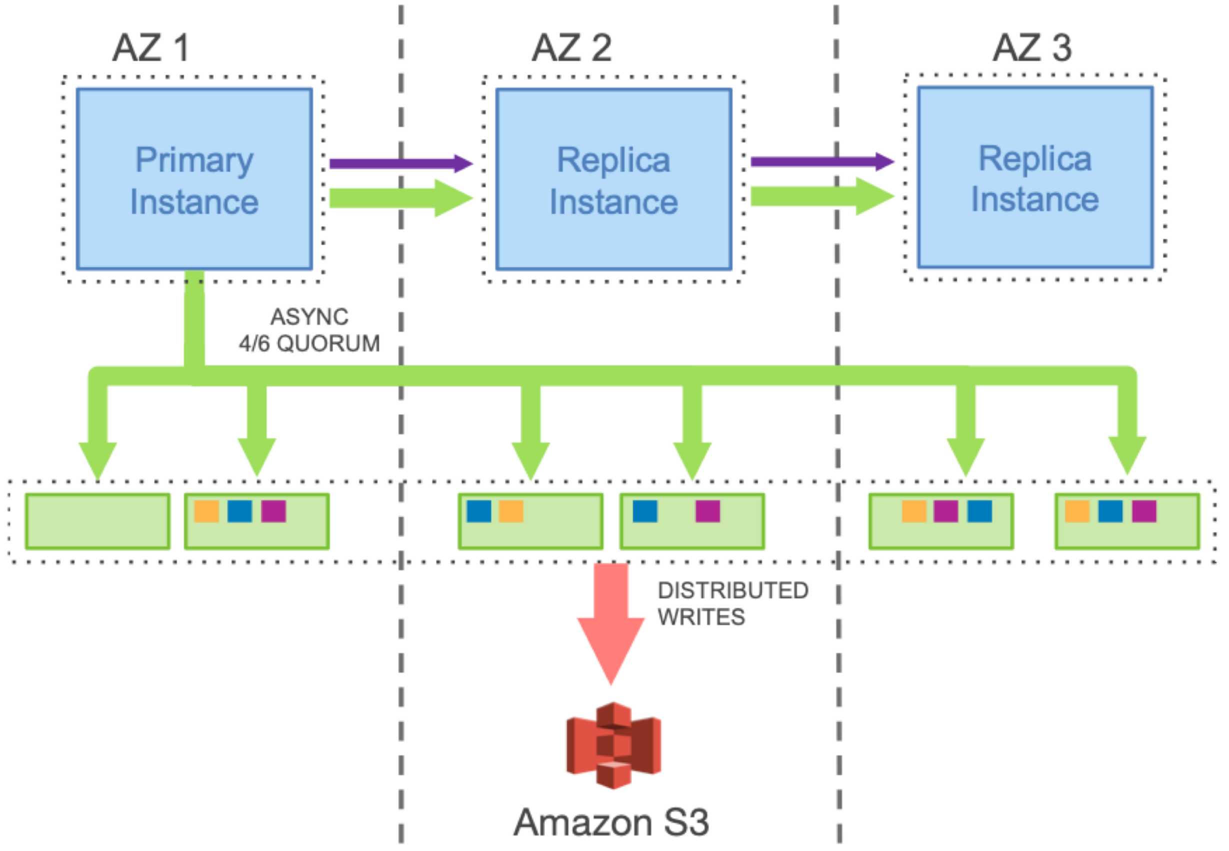


Source: *Amazon Aurora: Design Considerations for High Throughput Cloud-Native Relational Databases*, SIGMOD 2017

Single-Master Replication — Active-Passive

Amazon Aurora

- Single primary instance, multiple replica instances
- Only primary instance serves read-write transactions
- Replica instances service readonly transactions

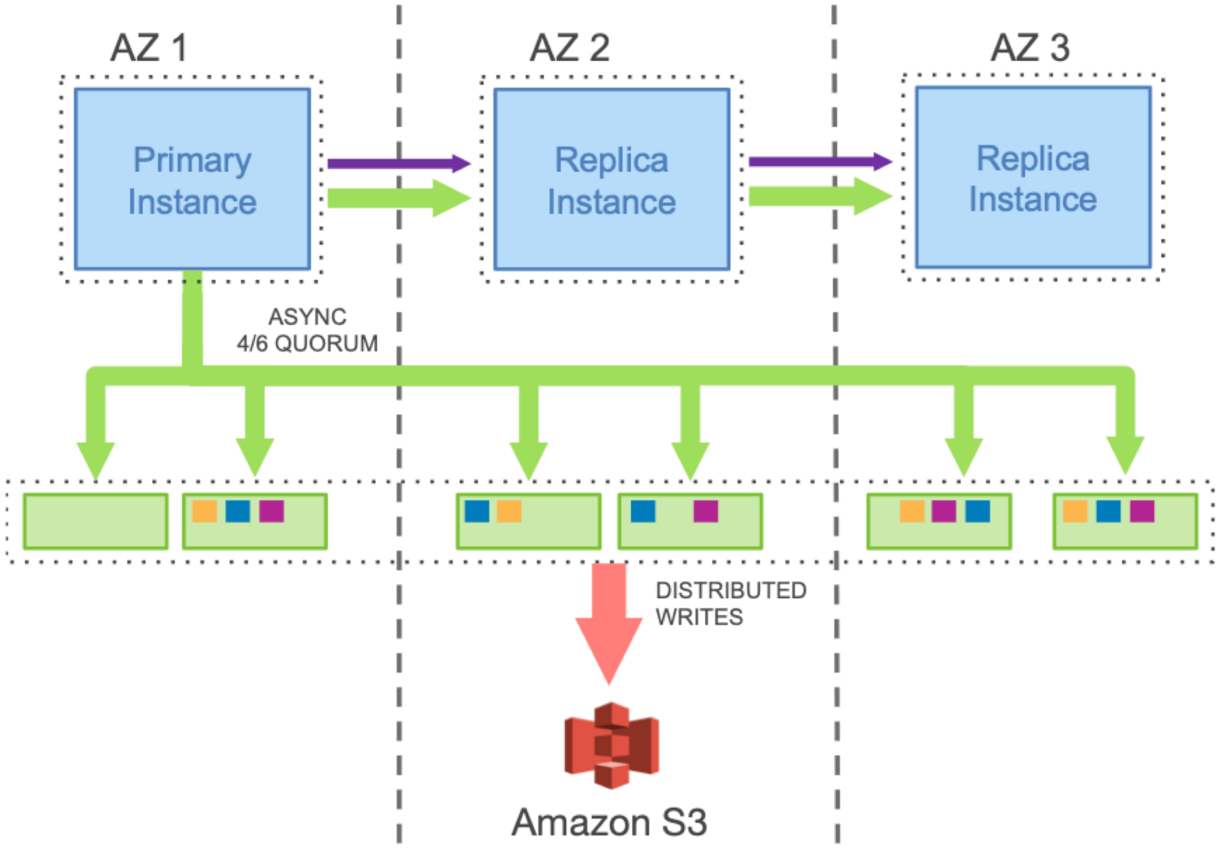


Source: *Amazon Aurora: Design Considerations for High Throughput Cloud-Native Relational Databases*, SIGMOD 2017

Single-Master Replication — Active-Passive

Amazon Aurora

- Single primary instance, multiple replica instances
- Only primary instance serves read-write transactions
- Replica instances service readonly transactions
- Updates replicated to replicas through **log shipping**
- **No locking performed in backup replicas**

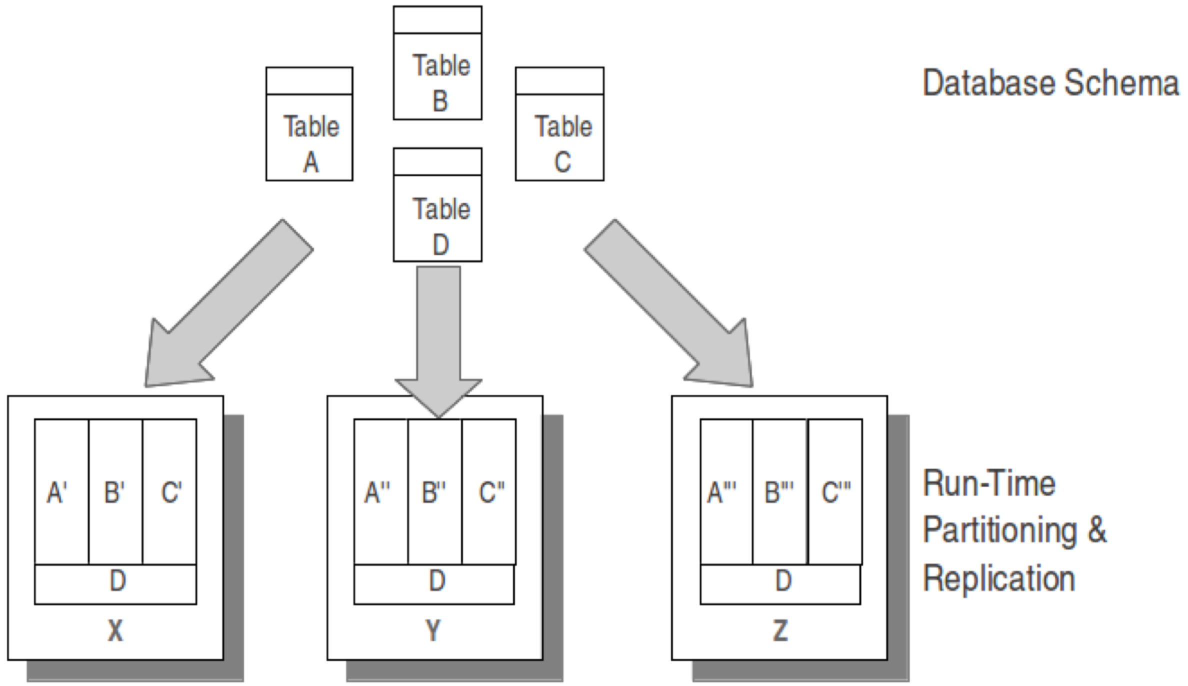


Source: *Amazon Aurora: Design Considerations for High Throughput Cloud-Native Relational Databases*, SIGMOD 2017

Single-Master Replication — Active-Active

VoltDB

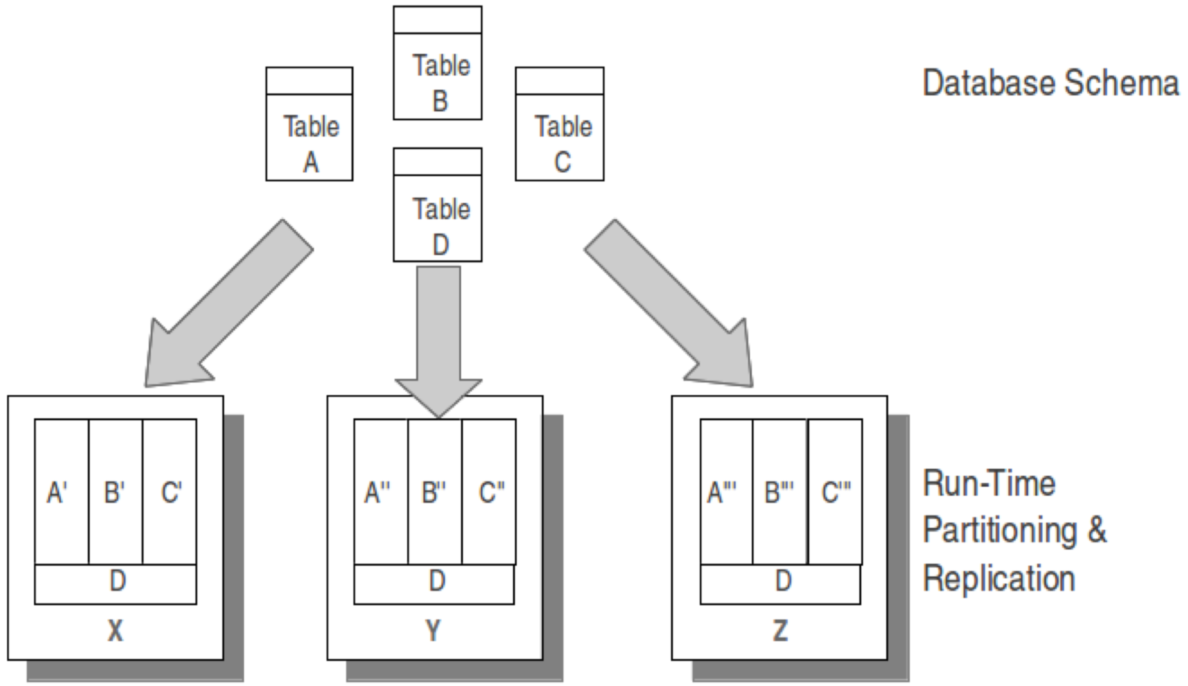
- All replicas are symmetric
- Each replica executes transaction sequentially. I.e., no concurrency control.



Single-Master Replication — Active-Active

VoltDB

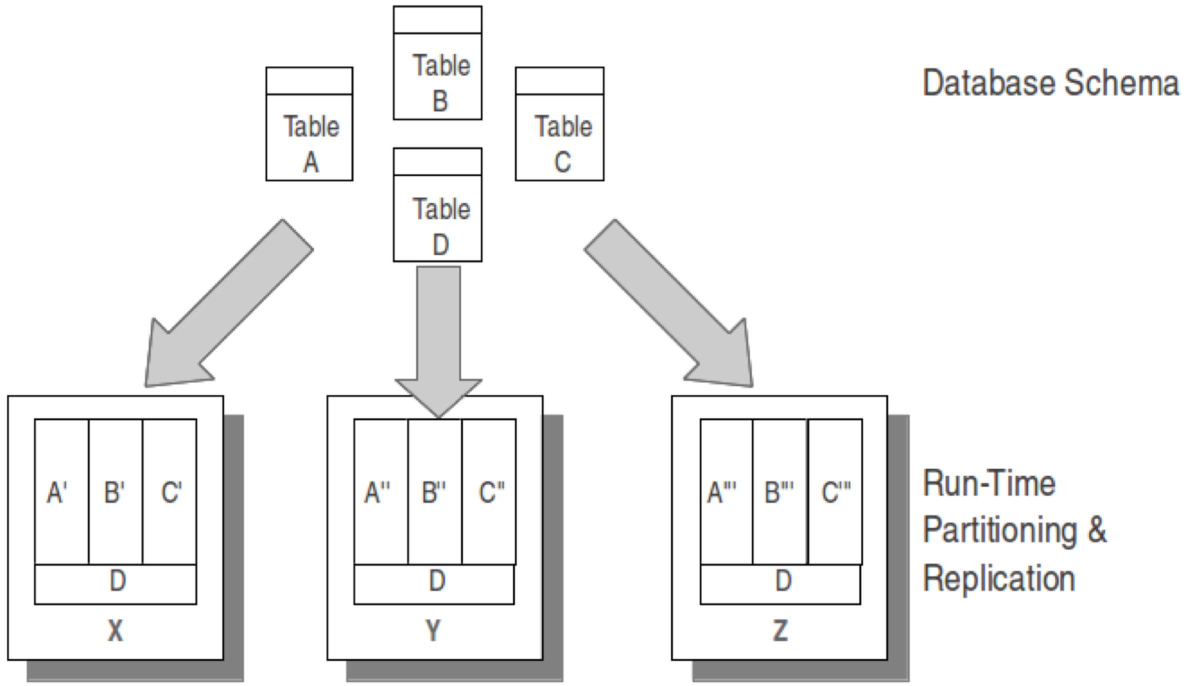
- All replicas are symmetric
- Each replica executes transaction sequentially. I.e., no concurrency control.
- **Key idea: determinism**. Each replica executes an identical sequence of transactions and produces identical updates



Single-Master Replication — Active-Active

VoltDB

- All replicas are symmetric
- Each replica executes transaction sequentially. I.e., no concurrency control.
- **Key idea: determinism**. Each replica executes an identical sequence of transactions and produces identical updates
- Only the transaction inputs (i.e., commands) need to be persistent and replicated, which happens before the transaction is executed

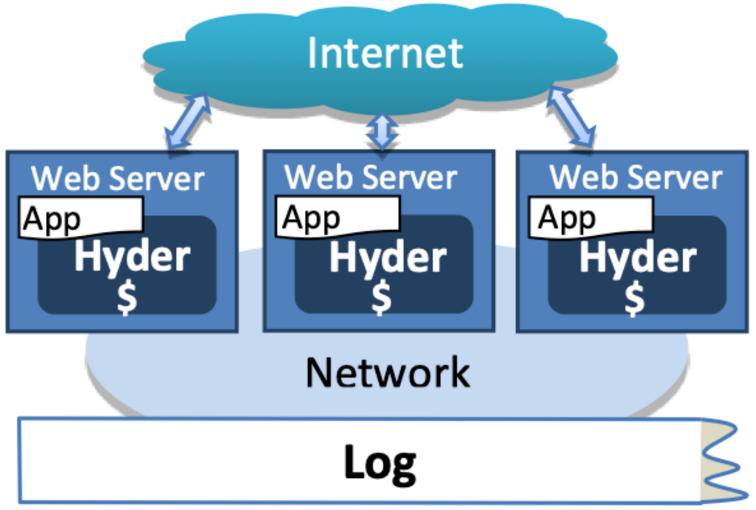


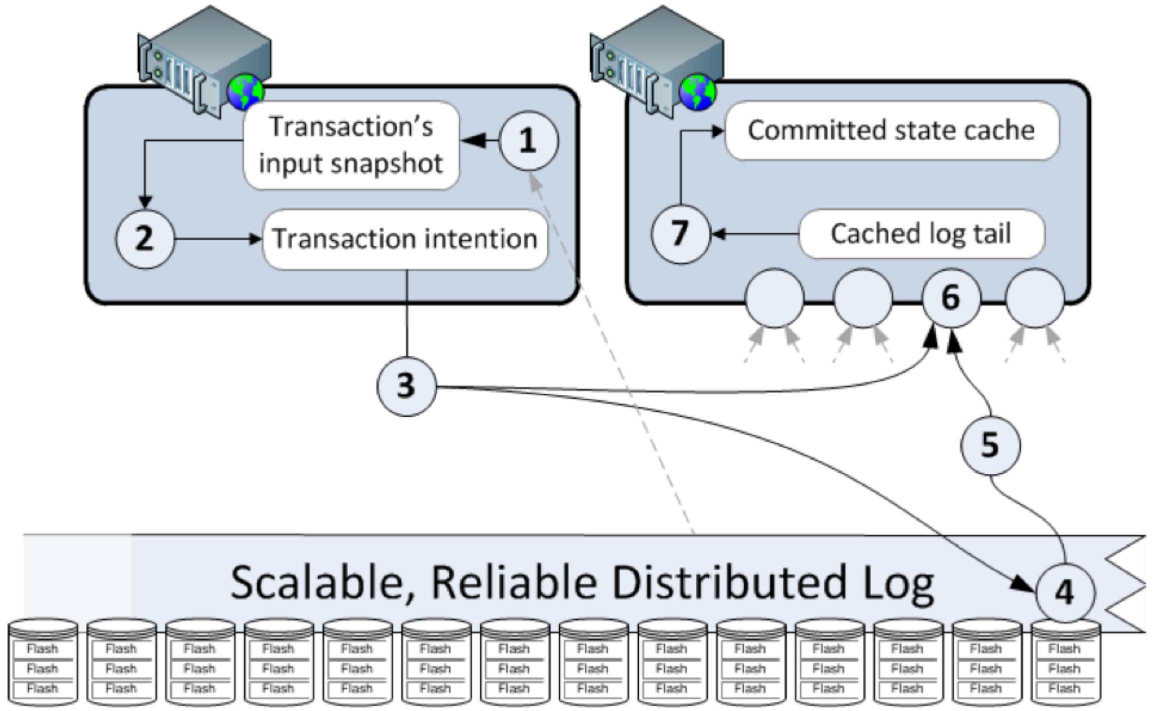
Q: How is this different from Multi-Master?

Multi-Master Replication

Hyder

- Can access all records in each server



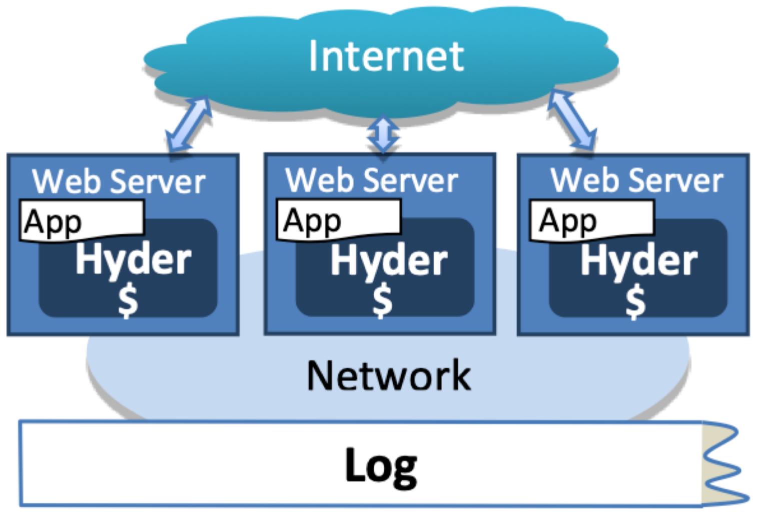


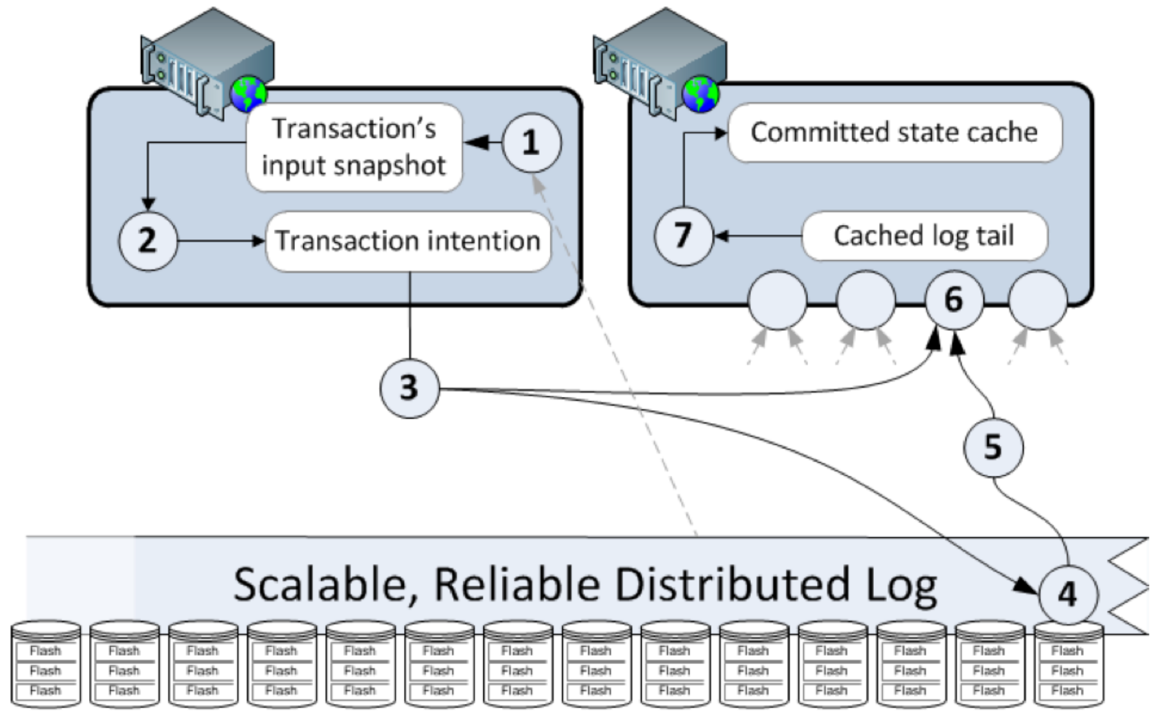
Source: *Hyder – A Transactional Record Manager for Shared Flash*, CIDR 2011

Multi-Master Replication

Hyder

- Can access all records in each server
- When local transaction finishes, write **intention** to a shared log



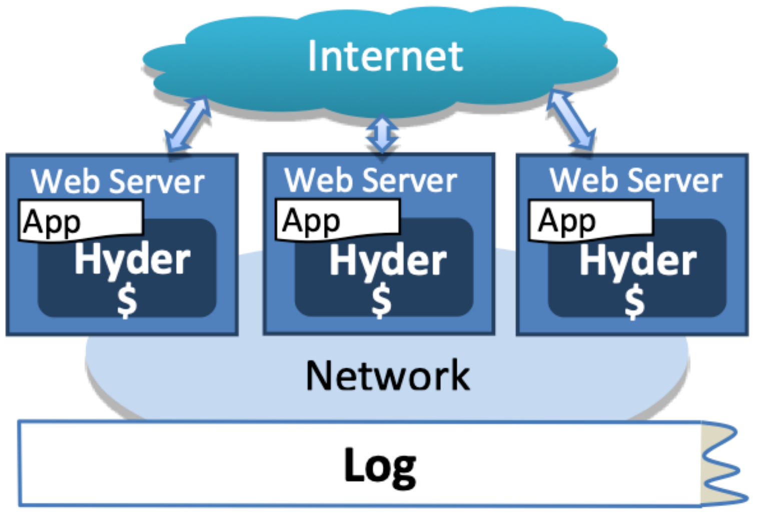


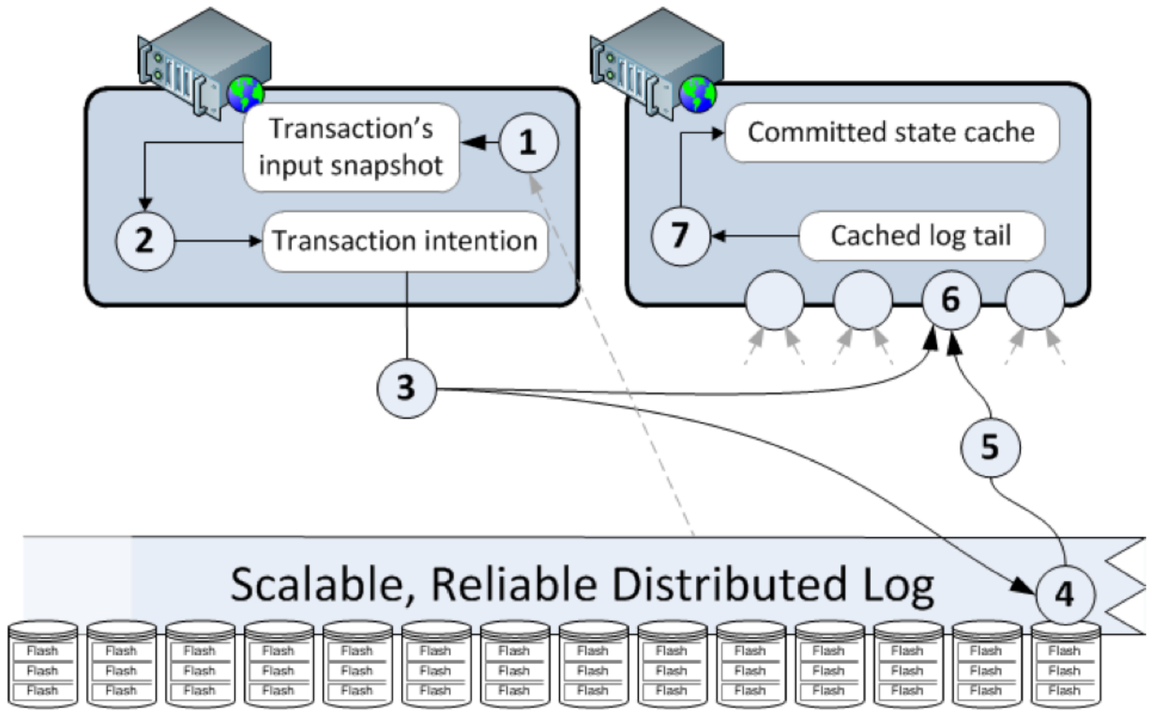
Source: *Hyder – A Transactional Record Manager for Shared Flash*, CIDR 2011

Multi-Master Replication

Hyder

- Can access all records in each server
- When local transaction finishes, write **intention** to a shared log
- Each server replays the shared log to detect conflicts and commit transactions





Source: *Hyder – A Transactional Record Manager for Shared Flash*, CIDR 2011

Revisit “The Danger of Replication”

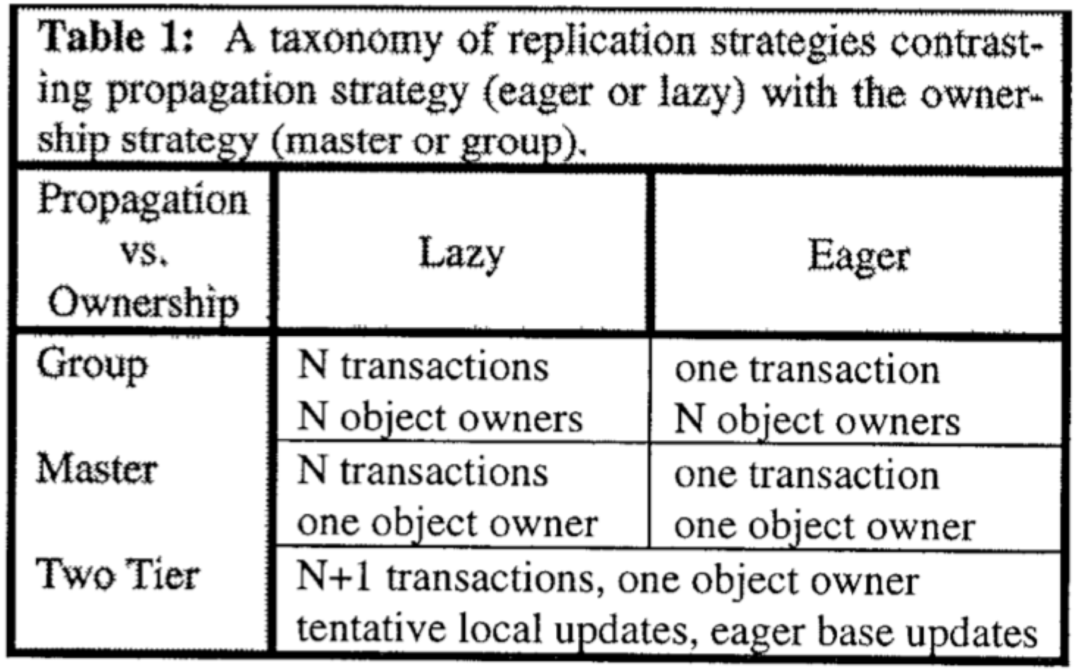
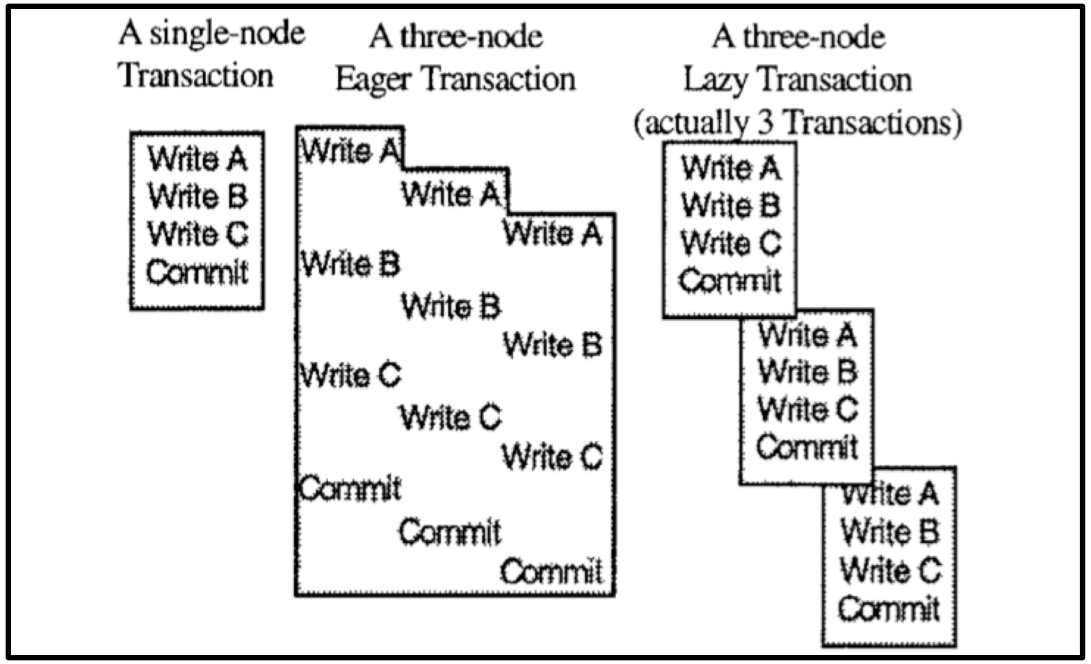
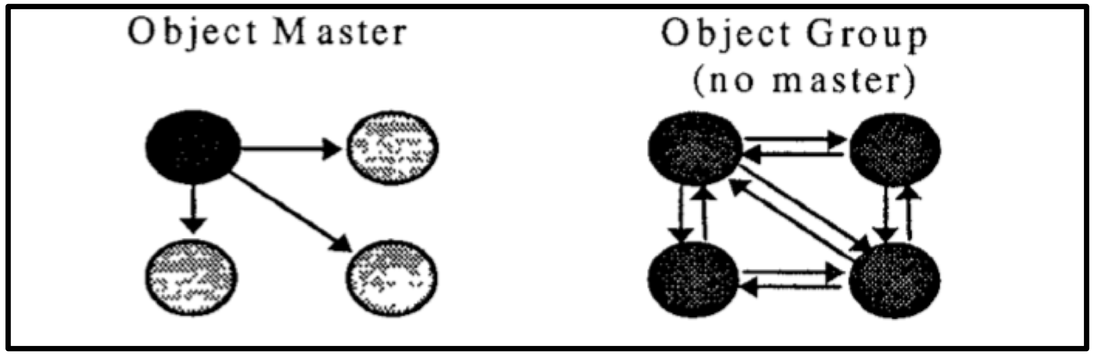


Table 1: A taxonomy of replication strategies contrasting propagation strategy (eager or lazy) with the ownership strategy (master or group).

Propagation vs. Ownership	Lazy	Eager
Group	N transactions N object owners	one transaction N object owners
Master	N transactions one object owner	one transaction one object owner
Two Tier	N+1 transactions, one object owner tentative local updates, eager base updates	



A single-node Transaction
A three-node Eager Transaction
A three-node Lazy Transaction (actually 3 Transactions)
Write A
Write B
Write C
Commit



Q: How do modern systems fit in this taxonomy?

Source: *Hyder – A Transactional Record Manager for Shared Flash*, CIDR 2011

Lazy vs. Eager

Lazy: commit the transaction before replication completes

- Typically sacrifice ACID
- Widely used in NoSQL systems

Lazy vs. Eager

Lazy: commit the transaction before replication completes

- Typically sacrifice ACID
- Widely used in NoSQL systems

Eager: must finish replication before transaction commits

- Can still use optimizations to reduce lock holding time
- E.g., Silo and Coco

Coco*

- Problem: 2PC and replication hurts throughput
  - Because transactions hold locks during these long-latency operations

* *Epoch-based Commit and Replication in Distributed OLTP Databases*, VLDB 2021

Coco*

Problem: 2PC and replication hurts throughput

– Because transactions hold locks during these long-latency operations

Key idea: Epoch-based commit and replication

– Commit a batch of transactions at a time (similar to Silo)

– Within an epoch, can release locks without waiting for logging or replication

* *Epoch-based Commit and Replication in Distributed OLTP Databases*, VLDB 2021

Coco*

Problem: 2PC and replication hurts throughput

– Because transactions hold locks during these long-latency operations

Key idea: Epoch-based commit and replication

– Commit a batch of transactions at a time (similar to Silo)

– Within an epoch, can release locks without waiting for logging or replication

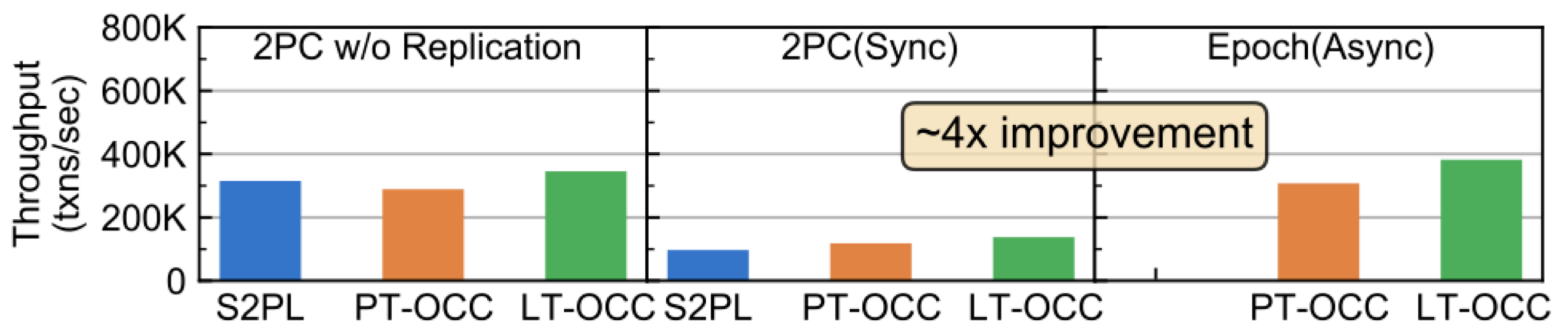


Figure 6: Throughput on TPC-C

* *Epoch-based Commit and Replication in Distributed OLTP Databases*, VLDB 2021

Ideal Replication Scheme

Availability and scalability: Provide high availability and scalability through replication, while avoiding instability

Mobility: Allow mobile nodes to read and update the database while disconnected from the network

- Modern distributed databases typically give up this property

Serializability: Provide single-copy serializable transaction execution

Convergence: Provide convergence to avoid system delusion

Two-Tier Replication

Mobile nodes are disconnected much of the time. They store a replica of the database and may originate tentative transactions. A mobile node may be the master of some data items.

- Submit tentative transactions to base nodes when connected

Base nodes are always connected. They store a replica of the database. Most items are mastered at base nodes

- Use lazy master replication

Modern systems implement two-tier replication in the application layer rather than the database layer

Q/A – Replication

How is replication related to 2PC? Use different commit protocol to update all replicas (same as eager?)

Relevance in modern systems?

What if the master fails?

Perform updates on replicas only when requesting data items?

Before Next Lecture

Submit review for

- Yi Lu, et al., Aria: A Fast and Practical Deterministic OLTP Database. VLDB, 2020