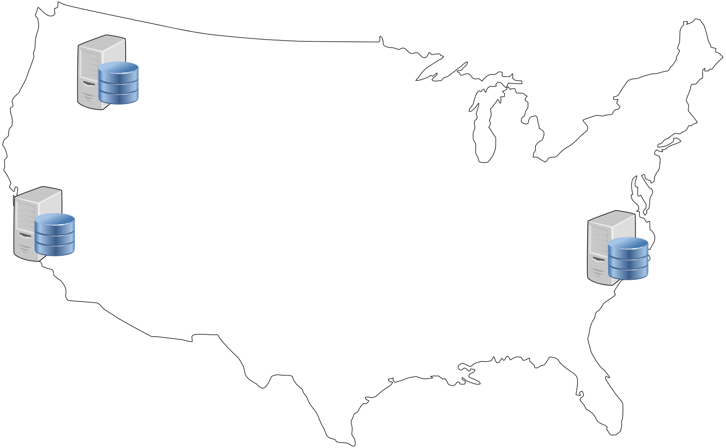
Geo-Replicated Transactions in 1.5RTT

Robert Escriva

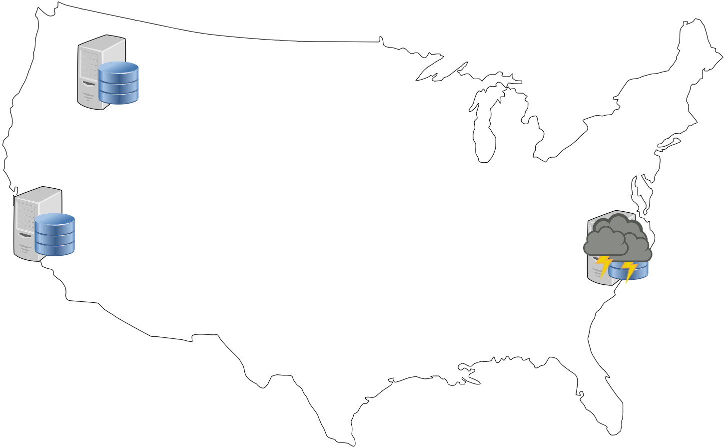
Strangeloop
September 30, 2017

Geo-Replication: A 539-Mile-High View



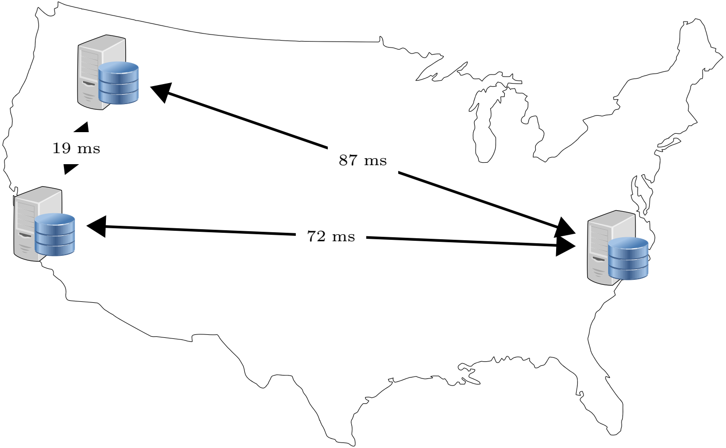
Geo-replicated distributed systems have servers in different data centers

Geo-Replication: A 539-Mile-High View



Failure of an entire data center is possible

Geo-Replication: A 539-Mile-High View



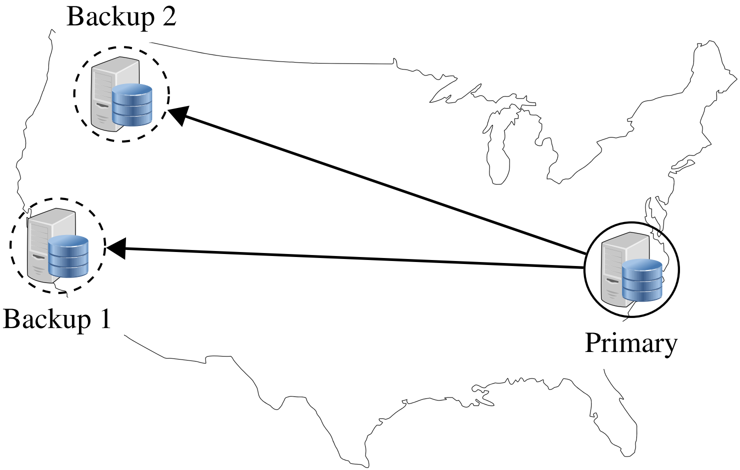
Latency between servers is on the order of tens to hundreds of milliseconds

Inter-Data Center Latency is Costly

In a geo-replicated system, latency is **the** dominating cost

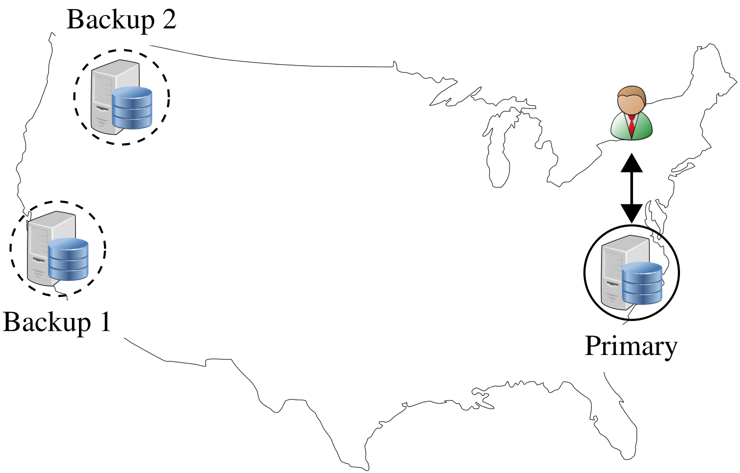
Memory Reference	100 ns	(100 ns)
4 kB SSD Read	150,000 ns	(150 µs)
Round Trip Same Data Center	500,000 ns	(500 µs)
HDD Disk Seek	8,000,000 ns	(8 ms)
Round Trip East-West	100,000,000 ns	(50 – 100 ms)

Geo-Replication: Primary Backup



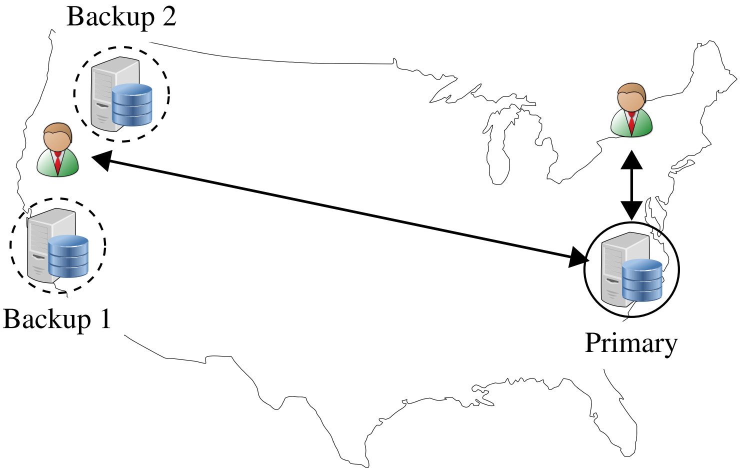
Writes happen at the primary and propagate to the backup

Geo-Replication: Primary Backup



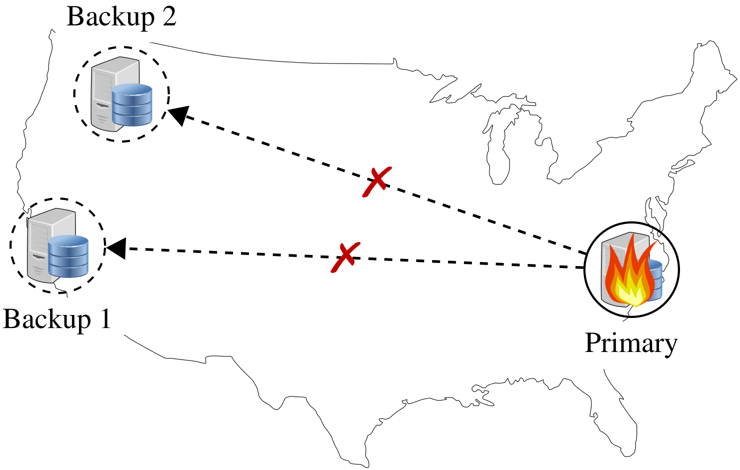
Clients close to the primary see low latency

Geo-Replication: Primary Backup



Clients close to a backup must still communicate with the primary

Geo-Replication: Primary Backup

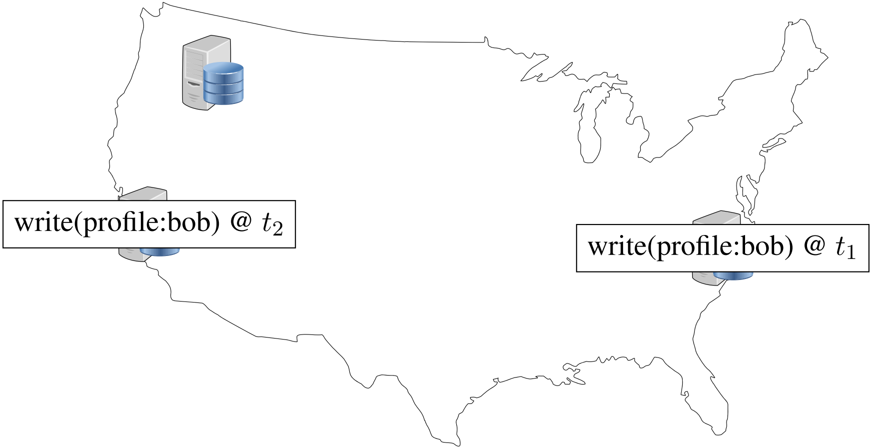


When the primary fails, operations stop until a new primary is selected

Primary/Backup

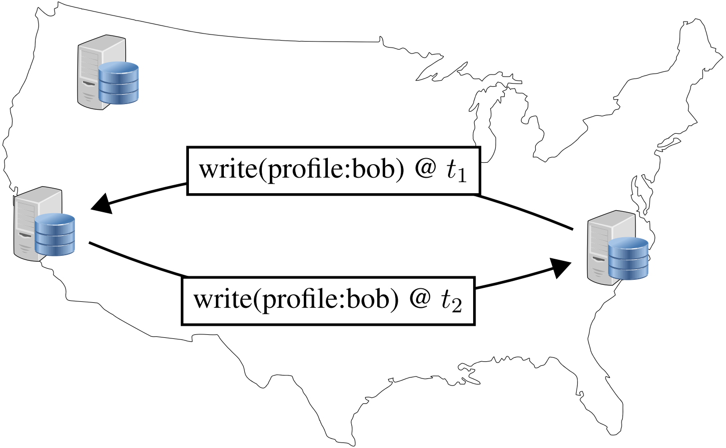
- ✓ Low-latency in the primary data center
- ✓ Simple to implement and reason about
- ✗ High-latency outside the primary data center
- ✗ Downtime during primary changeover

Geo-Replication: Eventual Consistency



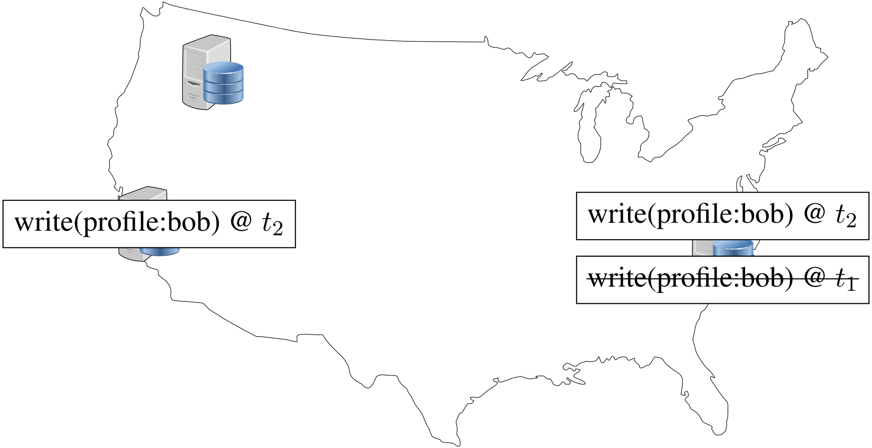
Eventually consistent systems write to each data center locally

Geo-Replication: Eventual Consistency



Writes eventually propagate between data centers

Geo-Replication: Eventual Consistency



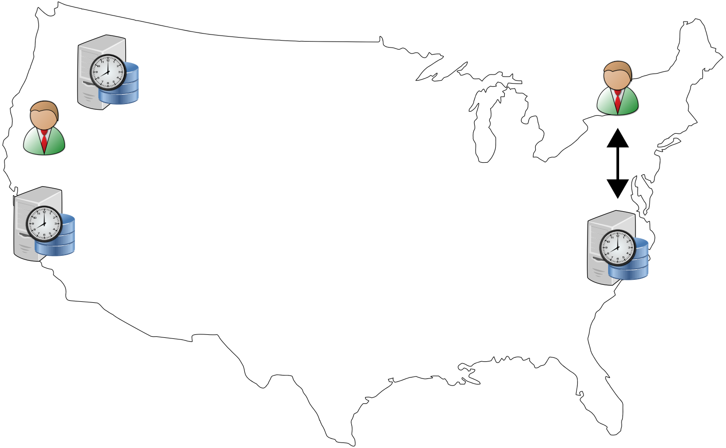
Concurrent writes may be lost—as if they never happened

Eventual Consistency

- ✓ Writes are always local and thus fast
- ✗ Data can be lost even if the write was successful
- ✓ Causal+-consistent systems with CRDTs will not lose writes
- ✗ But have no means of guaranteeing a read sees the “latest” value

Causal+ Consistency	Guarantees values converge to the same value using an associative and commutative merge function
Conflict-Free Replicated Data Types	Data structures that provide associative and commutative merge functions

Geo-Replication: TrueTime

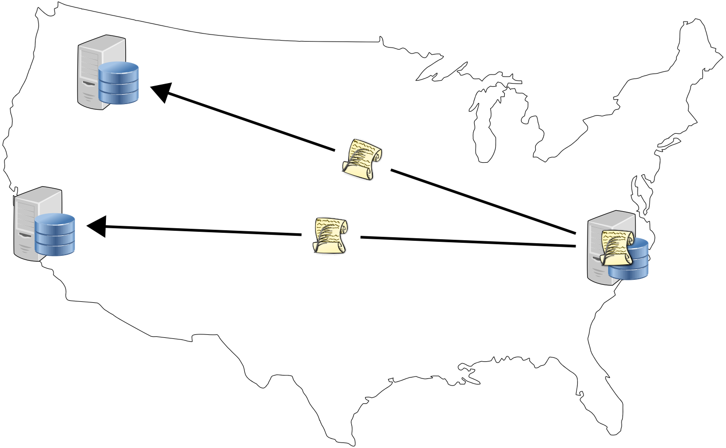


Synchronized clocks can enable efficient lockfree reads

Spanner and True Time

- ✓ Fast read-only transactions execute within a single data center
- Write path uses traditional 2-phase locking and 2-phase commit
- ✗ 2PL incurs cross-data center traffic during the body of the transaction (sometimes)

Geo-Replication: One-shot Transactions



One-shot transactions replicate the transaction input

Stored procedures and one-shot transactions

- Replicate the transaction, not its side effects
- Generally combined with commit protocol for scheduling
- ✓ Replicate the code, starting at any data center
- ✓ Succeeds in the absence of contention or failure
- ✗ Additional transactions may be required for fully general transactions

Consus Overview

Primary-less design	Applications contact the nearest data center
Serializable transactions	The gold standard in database guarantees
Efficient commit	Commit in 3 wide-area message delays

Consus Overview

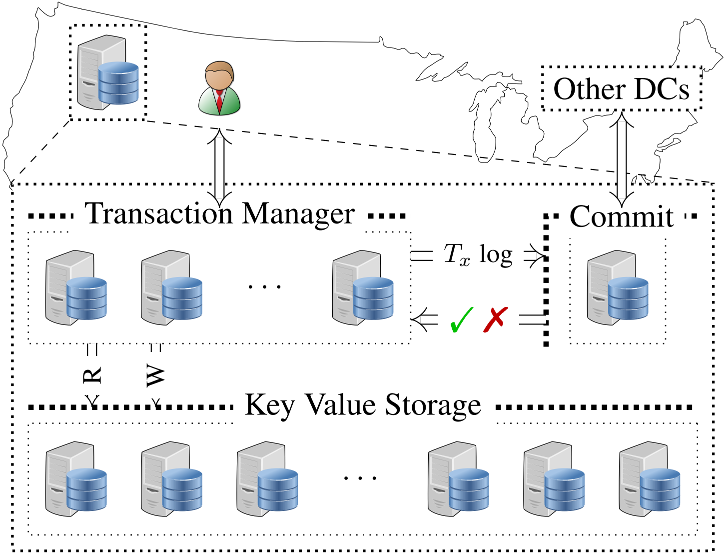
Primary-less design	Applications contact the nearest data center
Serializable transactions	The gold standard in database guarantees
Efficient commit	Commit in 3 wide-area message delays

Consus Contributions

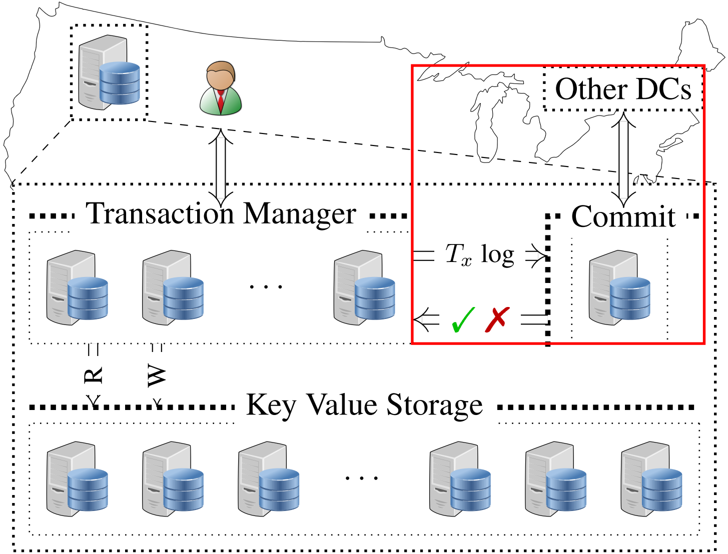
Consus' key contribution is a new commit protocol that:

- Executes transactions against a single data center
- Replays and decides transactions in 3 wide-area message delays
- Builds upon existing proven-correct consensus protocols

Geo-Replication: Consus



Geo-Replication: Consus



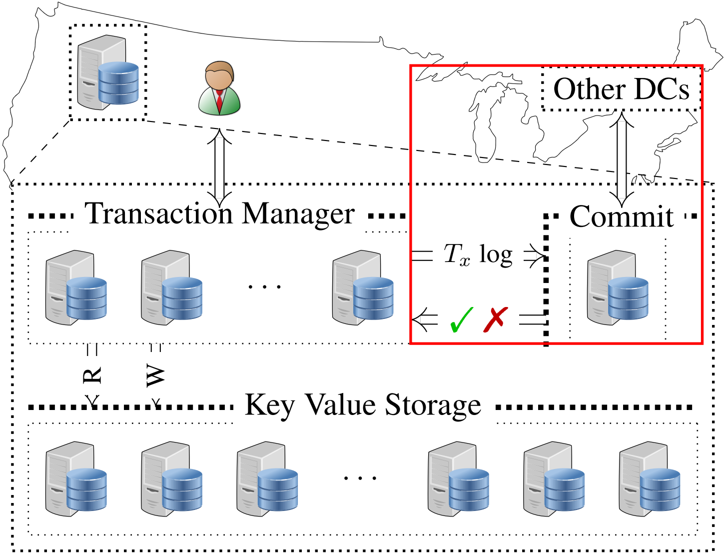
Commit Protocol Assumptions

- Each data center has a full replica of the data and a transaction processing engine
- The transaction processor is capable of executing a transaction up to the `prepare` stage of two-phase commit
- The transaction processor will abide the results of the commit protocol

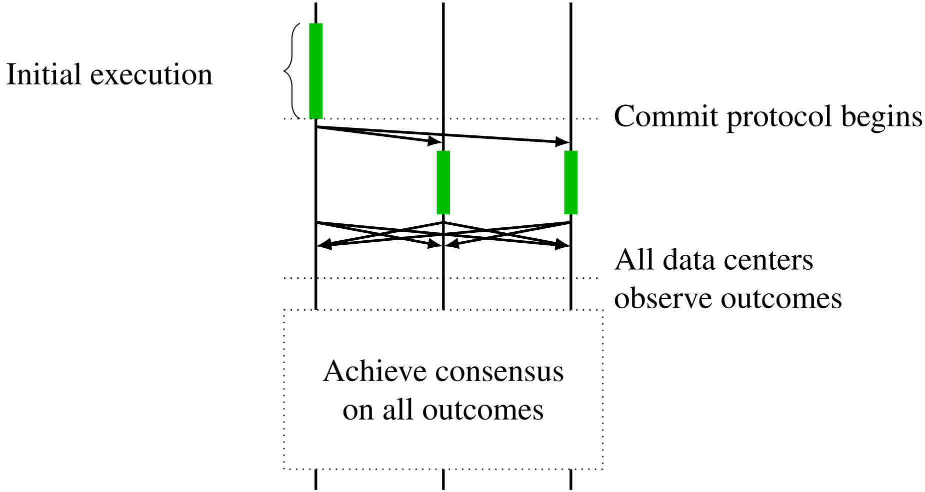
Commit Protocol Basics

- Transactions may commit if and only if a quorum of data centers can commit the transaction
- Transaction executes to “prepare” stage in one data center, and then executes to the “prepare” stage in every other data center
- The result of the commit protocol is binding
- Data centers that could not execute the transaction will enter degraded mode and synchronize the requisite data

Consus's Core Contribution



Overview of the Commit Protocol

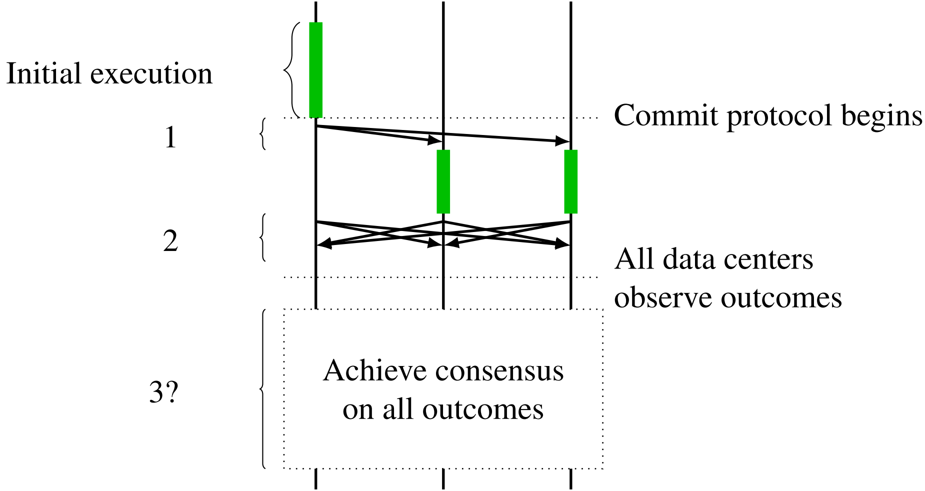


Observing vs. Learning Execution Outcomes

Why does Consus have a consensus step?

- A data center observing an outcome only knows that outcome
- Observation is insufficient to commit; another data center may not have yet made the same observation
- A data center learning an outcome knows that every non-faulty data center will learn the outcome
- The consensus step guarantees all (non-faulty) data centers can learn all outcomes

Counting Message Delays



Traditional Paxos

Paxos makes it possible to learn a value [Lam05]:

Nontriviality Any value learned must have been proposed

Stability A learn can learn at most one value

Consistency Two different learners cannot learn different values

Liveness If value C has been proposed, then eventually learner l will learn some value [1]

[1] This directly contradicts FLP. I'd be happy to reconcile the two after the talk.

Traditional Paxos

Paxos can be used to generate a sequence or log of values:

1. <Value chosen by Paxos_1>
2. <Value chosen by Paxos_2>
3. <Value chosen by Paxos_3>

   ...

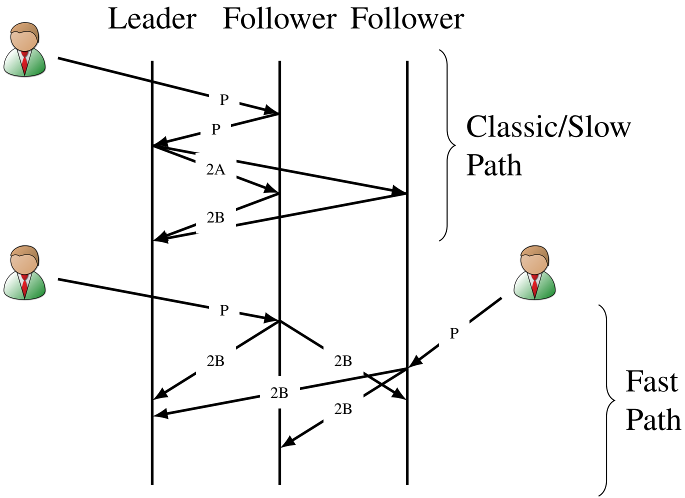
N. <Value chosen by Paxos_N>

Generalized Paxos

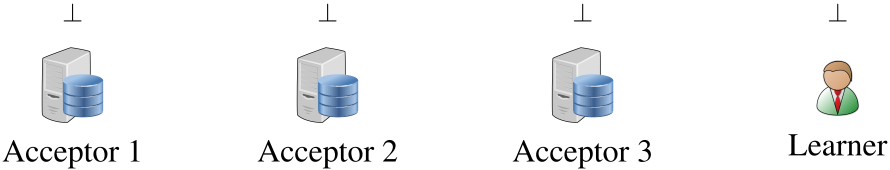
- Traditional Paxos agrees upon a sequence of values
  - View another way, Paxos agrees upon a totally ordered set
- Generalized Paxos agrees upon a partially ordered set
- Values learned by Gen. Paxos grow the partially ordered set incrementally, e.g. if a server learns v at t_1 and w at t_2, and $t_1 < t_2$, then $v \sqsubseteq w$

- Crucial property: Gen. Paxos has a fast path where acceptors can accept proposals without communicating with other acceptors

Generalized Paxos Fast Path

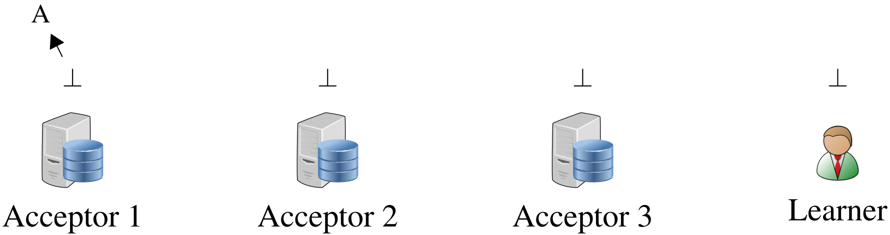


Generalized Paxos Example



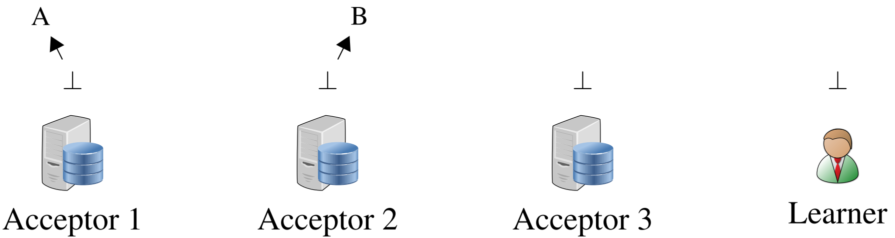
Initially all acceptors have an empty partially ordered set

Generalized Paxos Example



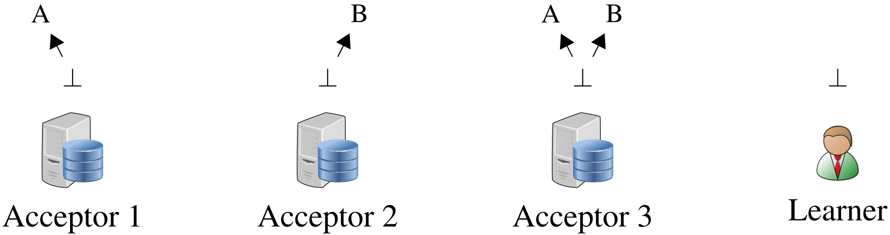
Acceptor 1 can accept “A” without consulting others

Generalized Paxos Example

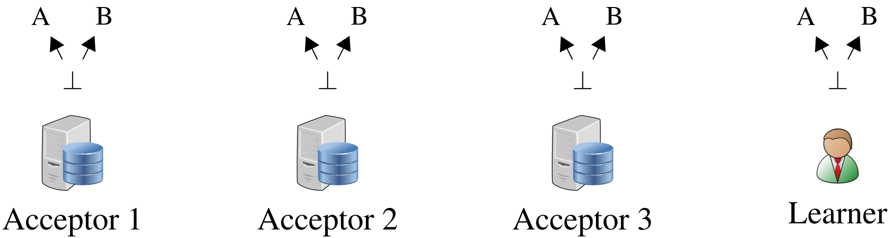


Acceptor 2 can accept “B” without consulting others

Generalized Paxos Example

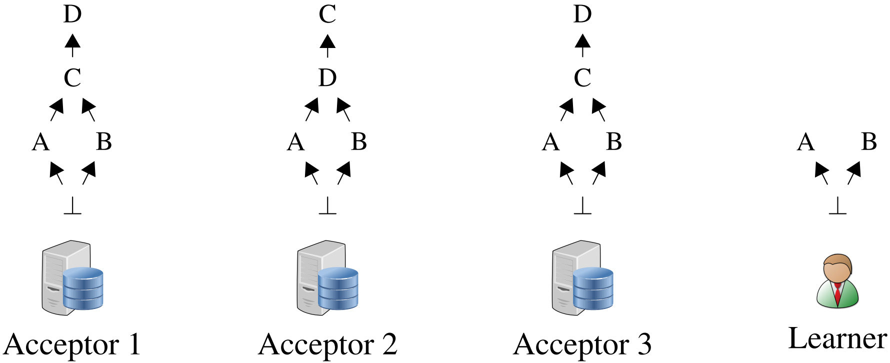


Generalized Paxos Example



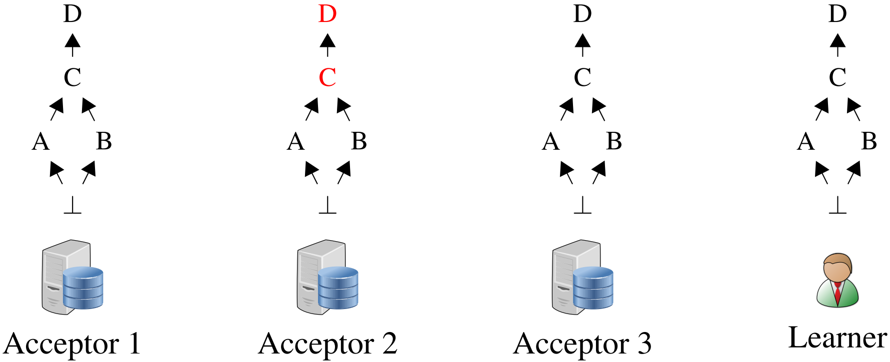
Only after a quorum accept “A” and “B” will the learner learn both

Generalized Paxos Example



When acceptors accept conflicting posets, a Classic round of Paxos is necessary

Generalized Paxos Example

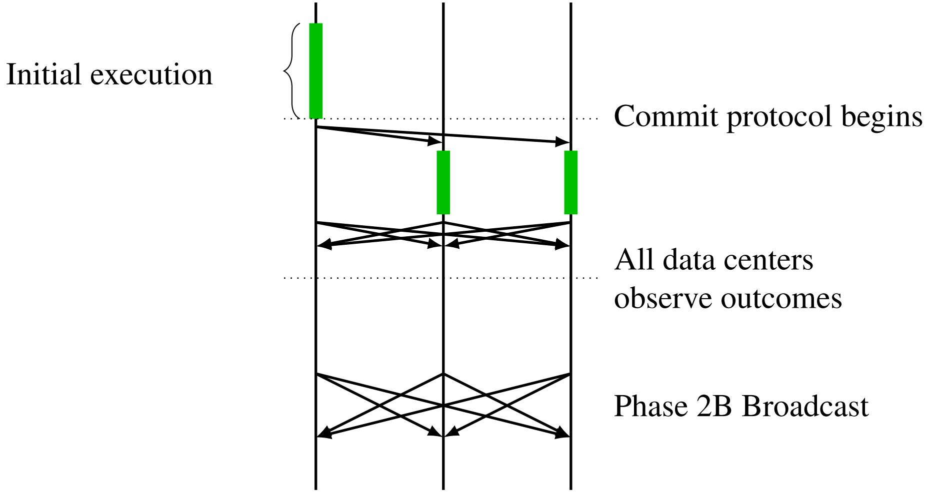


When acceptors accept conflicting posets, a Classic round of Paxos is necessary

Using Generalized Paxos in Consus

- Run one instance of Generalized Paxos per transaction
- Let the set of learnable commands be outcomes for the different data centers
- Outcomes are incomparable in acceptors' posets (effectively making them unordered sets)
- After accepting an outcome, broadcasting the newly accepted state
- Each data center's learner will eventually learn the same poset

Overview of the Commit Protocol



Cauterizing Loose Ends

Garbage Collection Generalized Paxos leaves garbage collection as an exercise for the reader

- Gen. Paxos instance lives only as long as a transaction
- Garbage collect entire instance, rather than part of poset

Deadlock Create a new command for a data center to request to change their outcome from “commit” to a “deadlock-induced abort”

- Totally order this with respect to all other commands
- May invoke slow path to abort a transaction

Performance Learning a poset requires checking equivalence relation and computing GLB for every possible quorum

- Pre-compute transitive closure of c-structs
- Use representation that is bit-wise operator friendly

Paxos All the Things!

Consus uses 4-5 different Paxos flavors/optimizations in its implementation:

- **Client-as-Leader:** Client holds a permanent ballot for transaction log
  - Transaction is fate shared with the client; optimizes away much of Paxos
- **Gray-Lamport Paxos Commit:** N instances of Paxos vote on commit
  - Each participant leads a round of Paxos to record its desire to commit or abort
- **Generalized Paxos:** Commit protocol described here-in
  - One acceptor per data center computes commit or abort for transaction
- **Recursive Paxos:** Each data center's acceptor is a Paxos RSM
  - Ensures a single server doesn't imply that a whole server has failed
- **Replicant** Replicated state machine hosting service
  - Write single-threaded code; run it in a fault-tolerant environment

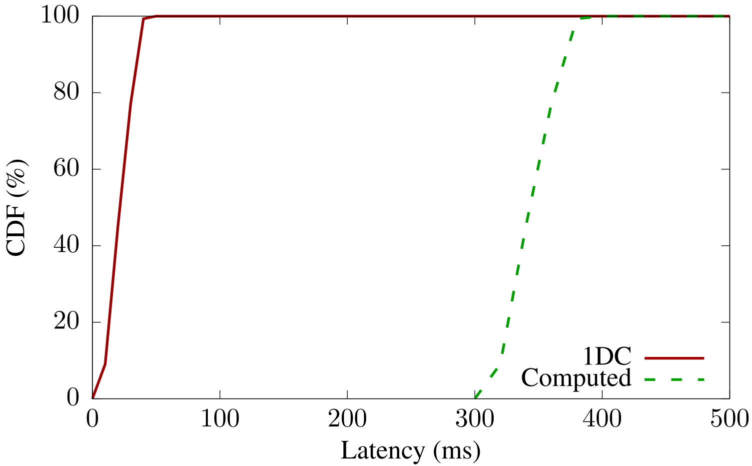
Implementation

- Approximately 32 k lines of code written for Consus and another 41 k imported from HyperDex dependencies
- Released under open source license
- Code is not production ready, but writes to disk and has the failure paths implemented

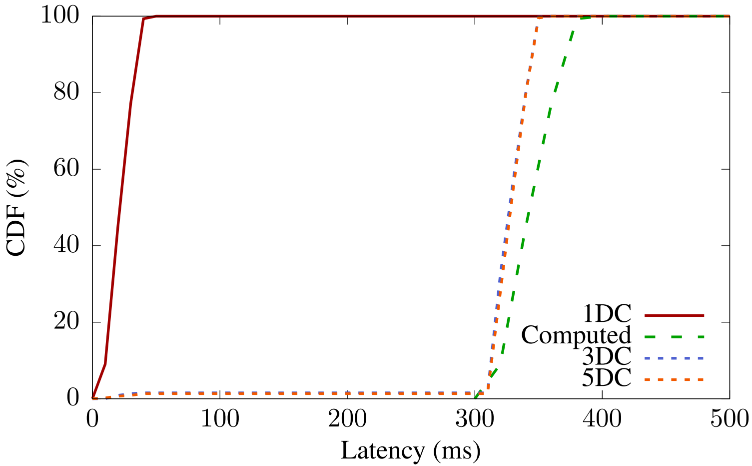
Evaluation Setup

- Experiments run on Amazon AWS using m3.xlarge instances with SSD storage
- Five servers deployed in the same availability zone
- Artificial RTT of 200 ms configured between servers to simulate wide-are setting
- One server for running TPC-C against the deployment

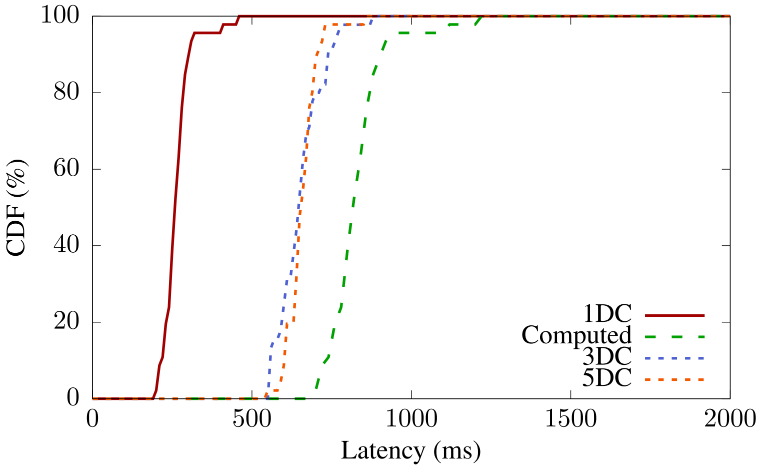
TPC-C New Order Latency



TPC-C New Order Latency



TPC-C Stock Level Latency



Summary

- Geo-replicated transactions can have lower latency
- Paxos is not a one-size-fits-all algorithm
- Careful specification of fault tolerance and availability requirements will guide your system's design

@rescrv

http://github.com/rescrv/

http://hack.systems/

Candidate Designs

- Primary/backup (often based on Paxos [Lam98])
  - Calvin [TDWR+12], Lynx [ZPZS+13], Megastore [BBCF+11], Rococco [MCZL+14], Scatter [GBKA11], Spanner [CDEF+13]
- Alternative consistency
  - Cassandra [LM09], CRDTs [SPBZ11], Dynamo [DHJK+07], $\mathcal{I}$-confluence analysis [BFFG+14], Gemini [LPCG+12], Walter [SPAL11]
- Spanner's TrueTime [CDEF+13]
  - Related: Granola [CL12], Loosely synchronized clocks [AGLM95]
- One-shot transactions
  - Janus [MNLL16], Calvin [TDWR+12], H-Store [KKNP+08], Rococco [MCZL+14]

Atul Adya, Robert Gruber, Barbara Liskov, and Umesh Maheshwari.
Efficient Optimistic Concurrency Control Using Loosely Synchronized Clocks.
In Proceedings of the SIGMOD International Conference on Management of Data, pages 23-34, San Jose, California, May 1995.

Peter Bailis, Alan Fekete, Michael J. Franklin, Ali Ghodsi, Joseph M. Hellerstein, and Ion Stoica.
Coordination-Avoiding Database Systems.
In CoRR, abs/1402.2237, 2014.

Jason Baker, Chris Bond, James C. Corbett, J. J. Furman, Andrey Khorlin, James Larson, Jean-Michel Leon, Yawei Li, Alexander Lloyd, and Vadim Yushprakh.
Megastore: Providing Scalable, Highly Available Storage For Interactive Services.

In Proceedings of the Conference on Innovative Data Systems Research, pages 223-234, Asilomar, California, January 2011.

James C. Corbett, Jeffrey Dean, Michael Epstein, Andrew Fikes, Christopher Frost, J. J. Furman, Sanjay Ghemawat, Andrey Gubarev, Christopher Heiser, Peter Hochschild, Wilson C. Hsieh, Sebastian Kanthak, Eugene Kogan, Hongyi Li, Alexander Lloyd, Sergey Melnik, David Mwaura, David Nagle, Sean Quinlan, Rajesh Rao, Lindsay Rolig, Yasushi Saito, Michal Szymaniak, Christopher Taylor, Ruth Wang, and Dale Woodford.
Spanner: Google's Globally Distributed Database.
In ACM Transactions on Computer Systems, 31(3):8, 2013.

James Cowling and Barbara Liskov.
Granola: Low-Overhead Distributed Transaction Coordination.
In Proceedings of the USENIX Annual Technical Conference, 2012.

- Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall, and Werner Vogels.
  Dynamo: Amazon's Highly Available Key-Value Store.
  In Proceedings of the Symposium on Operating Systems Principles, pages 205-220, Stevenson, Washington, October 2007.
- Lisa Glendenning, Ivan Beschastnikh, Arvind Krishnamurthy, and Thomas E. Anderson.
  Scalable Consistency In Scatter.
  In Proceedings of the Symposium on Operating Systems Principles, pages 15-28, Cascais, Portugal, October 2011.

Robert Kallman, Hideaki Kimura, Jonathan Natkins, Andrew Pavlo, Alex Rasin, Stanley B. Zdonik, Evan P. C. Jones, Samuel Madden, Michael Stonebraker, Yang Zhang, John Hugg, and Daniel J. Abadi.
H-Store: A High-Performance, Distributed Main Memory Transaction Processing System.
In Proceedings of the VLDB Endowment, 1(2):1496-1499, 2008.

Avinash Lakshman and Prashant Malik.
Cassandra: A Decentralized Structured Storage System.
In Proceedings of the International Workshop on Large Scale Distributed Systems and Middleware, Big Sky, Montana, October 2009.

Leslie Lamport.
Generalized Consensus And Paxos.
Microsoft Research, Technical Report MSR-TR-2005-33, 2005.

Leslie Lamport.
The Part-Time Parliament.
In ACM Transactions on Computer Systems, 16(2):133-169, 1998.

Cheng Li, Daniel Porto, Allen Clement, Johannes Gehrke, Nuno M. Preguiça, and Rodrigo Rodrigues.
Making Geo-Replicated Systems Fast As Possible, Consistent When Necessary.
In Proceedings of the Symposium on Operating System Design and Implementation, pages 265-278, Hollywood, California, October 2012.

Shuai Mu, Yang Cui, Yang Zhang, Wyatt Lloyd, and Jinyang Li.
Extracting More Concurrency From Distributed Transactions.
In Proceedings of the Symposium on Operating System Design and Implementation, pages 479-494, Broomfield, Colorado, October 2014.

Shuai Mu, Lamont Nelson, Wyatt Lloyd, and Jinyang Li.
Consolidating Concurrency Control And Consensus For Commits Under Conflicts.
In Proceedings of the Symposium on Operating System Design and Implementation, pages 517-532, Savannah, Georgia, November 2016.

Marc Shapiro, Nuno M. Preguiça, Carlos Baquero, and Marek Zawirski.
Conflict-Free Replicated Data Types.
In Proceedings of the Stabilization, Safety, and Security of Distributed Systems, pages 386–400, Grenoble, France, October 2011.

Yair Sovran, Russell Power, Marcos K. Aguilera, and Jinyang Li.
Transactional Storage For Geo-Replicated Systems.
In Proceedings of the Symposium on Operating Systems Principles, pages 385-400, Cascais, Portugal, October 2011.

- Alexander Thomson, Thaddeus Diamond, Shu-Chun Weng, Kun Ren, Philip Shao, and Daniel J. Abadi.
  Calvin: Fast Distributed Transactions For Partitioned Database Systems.
  In Proceedings of the SIGMOD International Conference on Management of Data, pages 1-12, Scottsdale, Arizona, May 2012.
- Yang Zhang, Russell Power, Siyuan Zhou, Yair Sovran, Marcos K. Aguilera, and Jinyang Li.
  Transaction Chains: Achieving Serializability With Low Latency In Geo-Distributed Storage Systems.
  In Proceedings of the Symposium on Operating Systems Principles, pages 276-291, 2013.