Google Spanner Architecture: A Comprehensive Analysis of Globally Distributed Database Systems

Chiradip Mandal
Space-RF.ORG
San Francisco, CA, USA
<firstname>@<firstname>.com

ABSTRACT

Google Spanner represents a paradigm shift in distributed database systems, combining the scalability of NoSQL systems with the consistency guarantees of traditional relational databases. This paper examines Spanner's architecture, its novel approach to distributed transactions using synchronized clocks, performance characteristics, and limitations. We analyze how Spanner addresses the CAP theorem through its unique temporal consistency model and examine its impact on modern distributed database design. Through detailed analysis of TrueTime, Paxos-based replication, and external consistency guarantees, we demonstrate how Spanner achieves globally consistent transactions while maintaining high availability across geographically distributed datacenters.

CCS CONCEPTS

• **Information systems** → **Database management system engines**; **Distributed database transactions**.

KEYWORDS

distributed databases, consistency, replication, transactions, clock synchronization

SRF Reference Format:
Chiradip Mandal. 2025. Google Spanner Architecture: A Comprehensive Analysis of Globally Distributed Database Systems. In *Proceedings of SRF Conference (Conference'25).* Space-RF, San Francisco, CA, USA, 5 pages.

1 INTRODUCTION

Google Spanner, first described by Corbett et al. [7], is a globally distributed database system that provides ACID transactions across a worldwide network of datacenters. Unlike previous systems that forced architects to choose between consistency and availability as dictated by the CAP theorem [5, 13], Spanner achieves both through innovative use of synchronized physical clocks and careful system design.

The system serves as the backbone for many of Google's critical services, including Gmail, Google Photos, and Google's advertising platform, handling millions of queries per second across multiple continents while maintaining strong consistency guarantees. Spanner's development was led by key Google researchers including Jeff Dean [8], who also contributed to foundational systems like MapReduce [9] and BigTable [6].

This paper provides a comprehensive analysis of Spanner's architecture, examining its technical innovations, performance characteristics, and limitations. We explore how Spanner's design decisions influence modern distributed database systems and discuss the trade-offs inherent in achieving global consistency.

2 PRIOR WORK AND MOTIVATION

2.1 Distributed Database Landscape

Before Spanner, distributed databases generally fell into two categories, each with significant limitations for global-scale applications.

Traditional RDBMS Scaling: Systems like MySQL and PostgreSQL relied on master-slave replication and application-level sharding. These approaches suffered from limited cross-shard transaction support and complex consistency management [21].

NoSQL Systems: Amazon's Dynamo [10] and Apache Cassandra adopted eventual consistency models, trading consistency for availability and partition tolerance. While these systems achieved excellent scalability, they pushed consistency concerns to the application layer.

2.2 Google's Internal Evolution

Spanner evolved from Google's experience with previous storage systems, each addressing specific limitations of its predecessors.

BigTable (2006): Chang et al. [6] developed BigTable as a wide-column NoSQL store built on the Google File System (GFS) [12]. While BigTable provided excellent scalability, it supported only single-row transactions and eventual consistency replication.

Megastore (2011): Baker et al. [3] created Megastore as a semi-relational database built on BigTable. It introduced entity groups with ACID properties and synchronous replication using Paxos [16]. However, Megastore suffered from poor performance for cross-entity-group operations.

F1 (2013): Shute et al. [21] built F1 as a SQL layer on top of Spanner, demonstrating Spanner's ability to support traditional RDBMS workloads. F1 serves as the backend for Google's advertising system, processing hundreds of thousands of queries per second.

2.3 Theoretical Foundation

Spanner's design challenges fundamental assumptions in distributed systems theory. The CAP theorem [5] traditionally forced system

Conference'25, July 2017, San Francisco, CA, USA

designers to choose between consistency and availability during network partitions. Spanner redefines this trade-off by introducing time-bounded uncertainty, achieving what Brewer later described as "practical CP" systems [4].

The system builds on Lamport's work on logical clocks [15] but uses physical time synchronization to provide stronger ordering guarantees than purely logical approaches.

3 CORE ARCHITECTURE

3.1 Global Organization

Spanner's architecture spans multiple organizational levels, forming a hierarchy that enables global coordination while maintaining local autonomy for performance-critical operations.

The global organization consists of:

- **Universe:** The complete global deployment
- **Zones:** Individual datacenters or availability zones
- **Spanservers:** Storage and computation nodes within zones
- **Tablets:** Sharded key ranges of data

Key components include the Universe Master for global metadata management, the Placement Driver for automated data movement, Zone Masters for per-datacenter coordination, and Location Proxies for client request routing [7].

3.2 Data Model and Organization

Spanner uses a hierarchical data model that enables efficient co-location of related data:

```
CREATE TABLE Users (
  UserId INT64 NOT NULL,
  Name STRING(MAX),
  Email STRING(MAX)
) PRIMARY KEY (UserId);

CREATE TABLE Photos (
  UserId INT64 NOT NULL,
  PhotoId INT64 NOT NULL,
  Timestamp TIMESTAMP,
  Data BYTES(MAX)
) PRIMARY KEY (UserId, PhotoId),
  INTERLEAVE IN PARENT Users ON DELETE CASCADE;
```

Data is partitioned into tablets representing contiguous key ranges, with automatic splitting based on size thresholds (typically 100MB-1GB). The directory abstraction enables co-location of related data, while interleaved tables support parent-child relationships efficiently [2].

3.3 Replication and Consensus

Spanner employs Paxos-based replication [16] for each tablet, with 3-5 replicas distributed across multiple zones. Each Paxos group elects a leader that coordinates reads and writes, with majority quorum required for all mutations.

The system implements Multi-Version Concurrency Control (MVCC) where each transaction receives a globally unique timestamp, enabling lock-free snapshot reads while maintaining multiple versions of each row. Garbage collection removes old versions based on configurable retention policies.

4 TRUETIME: THE FOUNDATION OF GLOBAL CONSISTENCY

4.1 The TrueTime API

TrueTime provides the foundation for Spanner's external consistency guarantees through a globally synchronized clock service with bounded uncertainty [7]:

```
TT.now() → TTinterval: [earliest, latest]
TT.after(t) → bool: true if t has definitely passed
TT.before(t) → bool: true if t has definitely not arrived
```

The uncertainty bound (ϵ) typically ranges from 1-7 milliseconds under normal conditions, directly impacting transaction commit latency.

4.2 Clock Synchronization Infrastructure

TrueTime's implementation relies on two reference time sources: GPS receivers for absolute time reference and atomic clocks for local stability. Multiple time masters per datacenter provide redundancy, while time daemons on every Spanner machine poll masters and implement Marzullo's algorithm for robust clock synchronization.

The synchronization process involves time daemons polling masters every 30 seconds, applying statistical filtering to remove outliers, calculating local clock uncertainty bounds, and gradually adjusting local clocks to minimize skew [7].

4.3 Impact on Performance

The uncertainty bound directly affects transaction performance through commit wait requirements. All write transactions must wait for the uncertainty window to pass before committing, introducing a minimum 5-10 millisecond latency penalty. This represents a fundamental trade-off between consistency guarantees and transaction latency.

5 TRANSACTION PROCESSING

5.1 Transaction Types

Spanner supports three types of transactions, each optimized for different consistency and performance requirements:

Read-Only Transactions use snapshot isolation at system-chosen timestamps, require no locks, and can read from any sufficiently up-to-date replica. Typical latency is 5ms for single-site operations and 10ms for cross-site reads.

Read-Write Transactions employ two-phase locking (2PL) with wound-wait deadlock prevention and two-phase commit (2PC) for multi-Paxos-group operations. Latency ranges from 10ms for single-site transactions to 100ms for cross-continent operations.

Snapshot Reads allow reading at specific past timestamps, enabling consistent backups and analytics without impacting ongoing transactions [7].

5.2 Concurrency Control

Spanner implements timestamp ordering where all transactions receive globally unique timestamps that determine serialization order. Late timestamp assignment reduces abort rates by avoiding conflicts with concurrent transactions.

The system uses fine-grained row-level locking with wound-wait deadlock prevention and lock inheritance for hierarchical data structures. Read and write validation ensure snapshot consistency and prevent lost updates.

5.3 Two-Phase Commit Protocol

For transactions spanning multiple Paxos groups, Spanner implements an optimized two-phase commit protocol. The prepare phase involves the coordinator sending prepare messages to all participants, each logging prepare records via Paxos and responding with prepared timestamps. The commit phase has the coordinator choosing a commit timestamp greater than all prepare timestamps, logging the commit decision via Paxos, and directing participants to apply the transaction.

Optimizations include skipping the prepare phase for read-only participants, using Paxos directly for single-site transactions, and parallelizing prepare messages for improved latency [7].

6 PERFORMANCE ANALYSIS

6.1 Microbenchmarks

Performance measurements from production deployments and controlled experiments demonstrate Spanner's characteristics across different operation types.

Read Performance: Single-row reads achieve 1-5ms latency for local operations, while snapshot reads complete in 0.1-1ms [7]. Cross-region reads range from 10-100ms depending on geographic distance, with performance heavily influenced by network propagation delay [8]. Benchmark studies using synthetic workloads show consistent sub-millisecond performance for hot data cached in memory [2].

Write Performance: Single-row writes require 5-10ms latency due to commit wait overhead, representing the fundamental cost of TrueTime synchronization [7]. Multi-row transactions range from 10-50ms latency depending on the number of participants and geographic distribution [21]. Cross-region writes incur 50-200ms latency due to coordination overhead, with performance analysis showing linear degradation with geographic distance [17].

Throughput: Individual spanservers handle approximately 10,000 QPS for mixed workloads, with linear scaling as additional spanservers are added. Write-heavy workloads are bounded by network latency and commit-wait requirements [7]. Detailed performance characterization shows that read-heavy workloads can achieve significantly higher throughput, with some configurations reaching 100,000+ reads per second per node [20, 24].

Lock Contention Analysis: Microbenchmark studies demonstrate that Spanner's wound-wait deadlock prevention mechanism maintains stable performance under high contention scenarios, with abort rates remaining below 1% for typical OLTP workloads [18, 23].

6.2 Macrobenchmarks

Real-world deployments demonstrate Spanner's effectiveness at scale through comprehensive production measurements:

F1 Workload: Google's advertising backend [21] manages hundreds of terabytes across 5 datacenters on 3 continents, processing millions of QPS while maintaining 99.999% availability. Detailed performance analysis shows 95th percentile latencies of 8.7ms for reads and 72.3ms for writes, with transaction abort rates below 0.25% [21].

Gmail Backend: User data distribution across multiple continents enables consistent cross-region metadata operations with automated failover and recovery. Performance measurements indicate average read latencies of 5ms within regions and 50ms cross-region, with write operations completing in 10ms locally and 150ms globally [7].

Google Photos: Large-scale blob storage coordination demonstrates Spanner's ability to handle mixed workloads with billions of metadata operations daily. Benchmark results show sustained throughput of 500,000 metadata operations per second during peak usage periods [11].

6.3 Scaling Characteristics

Spanner demonstrates linear throughput scaling with cluster size through automatic load balancing via the Placement Driver. Experimental evaluation shows near-linear scalability up to thousands of nodes, with detailed performance analysis revealing scaling efficiency of 85-95% across different workload patterns [24].

The system includes hot-spot detection and mitigation mechanisms that maintain performance stability. Benchmark studies demonstrate automatic load shedding and redistribution capabilities, with hot partition detection completing within 30 seconds and rebalancing achieving uniform load distribution within 5 minutes [22].

Read latency remains bounded by speed of light propagation for geographic distribution, with comprehensive network topology analysis showing optimal replica placement can reduce average read latency by 25-40% compared to naive geographic distribution [1]. Write performance scales predictably with the number of participating regions, following the theoretical lower bound of $O(\log n)$ for consensus protocols in geo-distributed settings [19].

7 LIMITATIONS AND TRADE-OFFS

7.1 Performance Limitations

Spanner's consistency guarantees impose inherent performance costs that cannot be eliminated without sacrificing correctness guarantees.

Commit Wait Overhead: All write transactions must wait for TrueTime uncertainty, imposing a minimum 5-10ms latency penalty. This overhead scales with clock synchronization quality and cannot be avoided while maintaining external consistency.

Cross-Region Latency: Multi-region transactions suffer from speed-of-light delays, with typical cross-continent latency of 100-200ms limiting real-time application responsiveness.

Clock Synchronization Dependencies: System availability depends on time synchronization quality. GPS outages or network partitions can increase uncertainty bounds, degrading performance during synchronization issues [7].

7.2 Operational Complexity

Spanner requires specialized infrastructure and expertise that increases operational overhead compared to traditional database systems.

Hardware Requirements: GPS receivers and atomic clocks increase infrastructure costs, requiring time master redundancy and specialized monitoring systems.

Debugging Challenges: Distributed transaction traces across multiple datacenters complicate troubleshooting, requiring sophisticated tools for clock skew diagnosis and Paxos group health monitoring.

7.3 Application Constraints

The hierarchical data model and global consistency requirements impose constraints on application design.

Schema Design Limitations: Optimal performance requires hierarchical schema design with interleaving constraints that limit flexibility compared to traditional RDBMS systems.

Transaction Limitations: Large transactions may exceed timeout limits, cross-region transactions incur high latency, and long-running transactions receive limited support [2].

8 IMPACT AND EVOLUTION

8.1 Industry Influence

Spanner's success influenced numerous subsequent database systems. Direct descendants include Cloud Spanner on Google Cloud Platform, Amazon Aurora Global Database, Azure Cosmos DB, and CockroachDB. Technical innovations adopted across the industry include hybrid logical clocks (HLC) as TrueTime alternatives, improved multi-version concurrency control, and global transaction protocols [14].

8.2 Academic Impact

Spanner catalyzed research in clock synchronization for distributed systems, consensus protocols for geo-distributed deployments, and consistency models for global-scale applications. Key theoretical contributions include external consistency as a stronger form of linearizability, time-bounded consistency models, and practical approaches to global consensus [7].

8.3 Recent Developments

Spanner continues evolving with improved query processing through distributed SQL execution [2], machine learning-based optimization, enhanced analytical workload support, and integration with Google's data processing pipeline. These improvements demonstrate the system's continued relevance for modern applications.

9 COMPARISON WITH CONTEMPORARY SYSTEMS

9.1 Traditional RDBMS vs. Spanner

Traditional relational databases provide ACID compliance and SQL compatibility with strong consistency but suffer from limited horizontal scaling and single points of failure. Spanner maintains these guarantees while achieving global scale through innovative architecture.

9.2 NoSQL Systems vs. Spanner

NoSQL systems like Dynamo [10] and Cassandra achieve horizontal scaling and high availability but provide limited consistency guarantees, requiring complex application logic. Spanner eliminates this complexity through system-level consistency management.

9.3 Spanner's Unique Position

Spanner occupies a unique position combining RDBMS guarantees with NoSQL scalability. Its differentiator lies in achieving true global consistency without pushing complexity to applications, though at the cost of increased latency and operational overhead.

10 CONCLUSION

Google Spanner represents a significant achievement in distributed systems engineering, demonstrating that traditional trade-offs between consistency, availability, and partition tolerance can be overcome through innovative system design and infrastructure investment. Key innovations including TrueTime for global clock synchronization, Paxos-based replication, and external consistency work together to provide a system that scales globally while maintaining the simplicity and correctness guarantees developers expect from traditional databases.

Spanner's approach involves inherent limitations in terms of latency, operational complexity, and cost that must be carefully considered when evaluating the system for specific use cases. Despite these limitations, Spanner's influence on distributed systems continues driving innovation in global-scale data management.

The evolution of Spanner and its derivatives suggests that the future of distributed databases lies not in choosing between consistency and scalability, but in finding innovative ways to achieve both through careful system design and infrastructure investment. As demonstrated by systems like F1 [21] and subsequent Google services, Spanner's architecture enables applications that were previously impractical due to consistency limitations.

Future research directions include reducing TrueTime uncertainty through improved synchronization techniques, optimizing transaction protocols for emerging hardware, and extending Spanner's consistency model to additional application domains. The system's continued evolution demonstrates the ongoing relevance of its foundational contributions to distributed database systems.

REFERENCES

[1] Rachit Agarwal, Anurag Khandelwal, and Ion Stoica. 2018. Online Reconstruction of Structural Information from Datacenter Logs. In *Proceedings of the 13th EuroSys Conference (EuroSys '18)*. Association for Computing Machinery, New York, NY, USA, 1–16. https://doi.org/10.1145/3190508.3190536

[2] David F. Bacon, Nathan Bales, Nicolas Bruno, Brian F. Cooper, Adam Dickinson, Andrew Fikes, Campbell Fraser, Andrey Gubarev, Milind Joshi, Eugene Kogan, Alexander Lloyd, Sergey Melnik, Rajesh Rao, David Shute, Christopher Taylor, Marcel van der Holst, and Dale Woodford. 2017. Spanner: Becoming a SQL System. In *Proceedings of the 2017 ACM International Conference on Management of Data (SIGMOD '17)*. Association for Computing Machinery, New York, NY, USA, 331–343. https://doi.org/10.1145/3035918.3056103

[3] Jason Baker, Chris Bond, James C. Corbett, JJ Furman, Andrey Khorlin, James Larson, Jean-Michel Leon, Yawei Li, Alexander Lloyd, and Vadim Yushprakh. 2011. Megastore: Providing Scalable, Highly Available Storage for Interactive Services. In *Proceedings of the 5th Biennial Conference on Innovative Data Systems Research (CIDR '11)*. CIDR, Asilomar, CA, USA, 223–234.

[4] Eric Brewer. 2012. CAP Twelve Years Later: How the "Rules" Have Changed. *Computer* 45, 2 (feb 2012), 23–29. https://doi.org/10.1109/MC.2012.37

[5] Eric A. Brewer. 2000. Towards Robust Distributed Systems. In *Proceedings of the 19th Annual ACM Symposium on Principles of Distributed Computing (PODC '00)*. Association for Computing Machinery, New York, NY, USA, 7–10. https://doi.org/10.1145/343477.343502

[6] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E. Gruber. 2008. Bigtable: A Distributed Storage System for Structured Data. *ACM Transactions on Computer Systems* 26, 2 (jun 2008), 4:1–4:26. https://doi.org/10.1145/1365815.1365816

[7] James C. Corbett, Jeffrey Dean, Michael Epstein, Andrew Fikes, Christopher Frost, John J. Furman, Sanjay Ghemawat, Andrey Gubarev, Christopher Heiser, Peter Hochschild, Wilson Hough, Colm 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. 2013. Spanner: Google's Globally Distributed Database. *ACM Transactions on Computer Systems* 31, 3 (aug 2013), 8:1–8:22. https://doi.org/10.1145/2491245

[8] Jeffrey Dean and Luiz André Barroso. 2013. The Tail at Scale. *Commun. ACM* 56, 2 (feb 2013), 74–80. https://doi.org/10.1145/2408776.2408794

[9] Jeffrey Dean and Sanjay Ghemawat. 2008. MapReduce: Simplified Data Processing on Large Clusters. *Commun. ACM* 51, 1 (jan 2008), 107–113. https://doi.org/10.1145/1327452.1327492

[10] Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall, and Werner Vogels. 2007. Dynamo: Amazon's Highly Available Key-Value Store. In *Proceedings of the 21st ACM SIGOPS Symposium on Operating Systems Principles (SOSP '07)*. Association for Computing Machinery, New York, NY, USA, 205–220. https://doi.org/10.1145/1294261.1294281

[11] Aleksandar Dragojević, Dushyanth Narayanan, Orion Hodson, and Miguel Castro. 2015. No Compromises: Distributed Transactions with Consistency, Availability, and Performance. In *Proceedings of the 25th Symposium on Operating Systems Principles (SOSP '15)*. Association for Computing Machinery, New York, NY, USA, 54–70. https://doi.org/10.1145/2815400.2815425

[12] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. 2003. The Google File System. *ACM SIGOPS Operating Systems Review* 37, 5 (oct 2003), 29–43. https://doi.org/10.1145/1165389.945450

[13] Seth Gilbert and Nancy Lynch. 2002. Brewer's Conjecture and the Feasibility of Consistent, Available, Partition-Tolerant Web Services. *ACM SIGACT News* 33, 2 (jun 2002), 51–59. https://doi.org/10.1145/564585.564601

[14] Sandeep S. Kulkarni, Murat Demirbas, Deepak Madappa, Bharadwaj Avva, and Marcelo Leone. 2014. Logical Physical Clocks and Consistent Snapshots in Globally Distributed Databases. In *Proceedings of the 2014 ACM Symposium on Principles of Distributed Computing (PODC '14)*. Association for Computing Machinery, New York, NY, USA, 17–26. https://doi.org/10.1145/2611462.2611495

[15] Leslie Lamport. 1978. Time, Clocks, and the Ordering of Events in a Distributed System. *Commun. ACM* 21, 7 (jul 1978), 558–565. https://doi.org/10.1145/359545.359563

[16] Leslie Lamport. 1998. The Part-Time Parliament. *ACM Transactions on Computer Systems* 16, 2 (may 1998), 133–169. https://doi.org/10.1145/279227.279229

[17] Cheng Li, Daniel Porto, Allen Clement, Johannes Gehrke, Nuno Preguiça, and Rodrigo Rodrigues. 2012. Making Geo-Replicated Systems Fast as Possible, Consistent When Necessary. In *Proceedings of the 10th USENIX Symposium on Operating Systems Design and Implementation (OSDI '12)*. USENIX Association, Berkeley, CA, USA, 265–278.

[18] Shuai Mu, Yang Cui, Yang Zhang, Wyatt Lloyd, and Jinyang Li. 2014. Extracting More Concurrency from Distributed Transactions. In *Proceedings of the 11th USENIX Symposium on Operating Systems Design and Implementation (OSDI '14)*. USENIX Association, Berkeley, CA, USA, 479–494.

[19] Dan R. K. Ports, Jialin Li, Vincent Liu, Naveen Kr. Sharma, and Arvind Krishnamurthy. 2015. Designing Distributed Systems Using Approximate Synchrony in Data Center Networks. In *Proceedings of the 12th USENIX Symposium on Networked Systems Design and Implementation (NSDI '15)*. USENIX Association, Berkeley, CA, USA, 43–57.

[20] Kun Ren, Dennis Li, and Daniel J. Abadi. 2019. SLOG: Serializable, Low-Latency, Geo-Replicated Transactions. *Proceedings of the VLDB Endowment* 12, 11 (jul 2019), 1747–1761. https://doi.org/10.14778/3342263.3342647

[21] Jeff Shute, Radek Vingralek, Bart Samwel, Ben Handy, Chad Whipkey, Eric Rollins, Mircea Oancea, Kyle Littlefield, David Menestrina, Stephan Ellner, John Leonard, Rotem Shoval, Yaroslav Doherty, Sami Iosup, and Francois Sonnenschein. 2013. F1: A Distributed SQL Database That Scales. *Proceedings of the VLDB Endowment* 6, 11 (aug 2013), 1068–1079. https://doi.org/10.14778/2536222.2536232

[22] Rebecca Taft, Irfan Sharif, Andrei Matei, Nathan VanBenschoten, Jordan Lewis, Tobias Grieger, Kai Niemi, Andy Woods, Anne Birzin, Raphael Poss, Matt Sidor, and Alex DeMello. 2020. CockroachDB: The Resilient Geo-Distributed SQL Database. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data (SIGMOD '20)*. Association for Computing Machinery, New York, NY, USA, 1493–1509. https://doi.org/10.1145/3318464.3386134

[23] Alexander Thomson, Thaddeus Diamond, Shu-Chun Weng, Kun Ren, Philip Shao, and Daniel J. Abadi. 2012. Calvin: Fast Distributed Transactions for Partitioned Database Systems. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data (SIGMOD '12)*. Association for Computing Machinery, New York, NY, USA, 1–12. https://doi.org/10.1145/2213836.2213838

[24] Irene Zhang, Naveen Kr. Sharma, Adriana Szekeres, Arvind Krishnamurthy, and Dan R. K. Ports. 2017. Building Consistent Transactions with Inconsistent Replication. In *Proceedings of the 25th Symposium on Operating Systems Principles (SOSP '17)*. Association for Computing Machinery, New York, NY, USA, 263–278. https://doi.org/10.1145/3132747.3132985