Modern Databases

Persistent storage in three flavors

26 March 2024

Dr. Jacob Hochstetler
Distinguished Engineer, Vice President, Fidelity Investments
Clinical Assistant Professor, University of North Texas

Agenda

- Introduction
- Background
- AWS' *Physalia*
- Cockroach Labs' *CockroachDB*
- TigerBeetle Inc.'s *TigerBeetle*
- Conclusion

Introduction

- **Evolution of databases:**
  - *Primitive* databases: Flat files and hierarchical models.
  - *Relational* databases: Introduction of SQL and normalization.
  - *NoSQL* shift: Adopting non-relational models for scalability.
  - *NewSQL* and distributed databases: Combining scalability with consistency.
- **Modern challenges:**

🌎Data volume/velocity: Manage big data and real-time processing globally.

📈Scalability: Monitor performance as data and user numbers grow.

💝High availability and disaster recovery: Minimize downtime and data loss.

👮Security/privacy: Avoid hackers, observe policies, and comply with regulations.

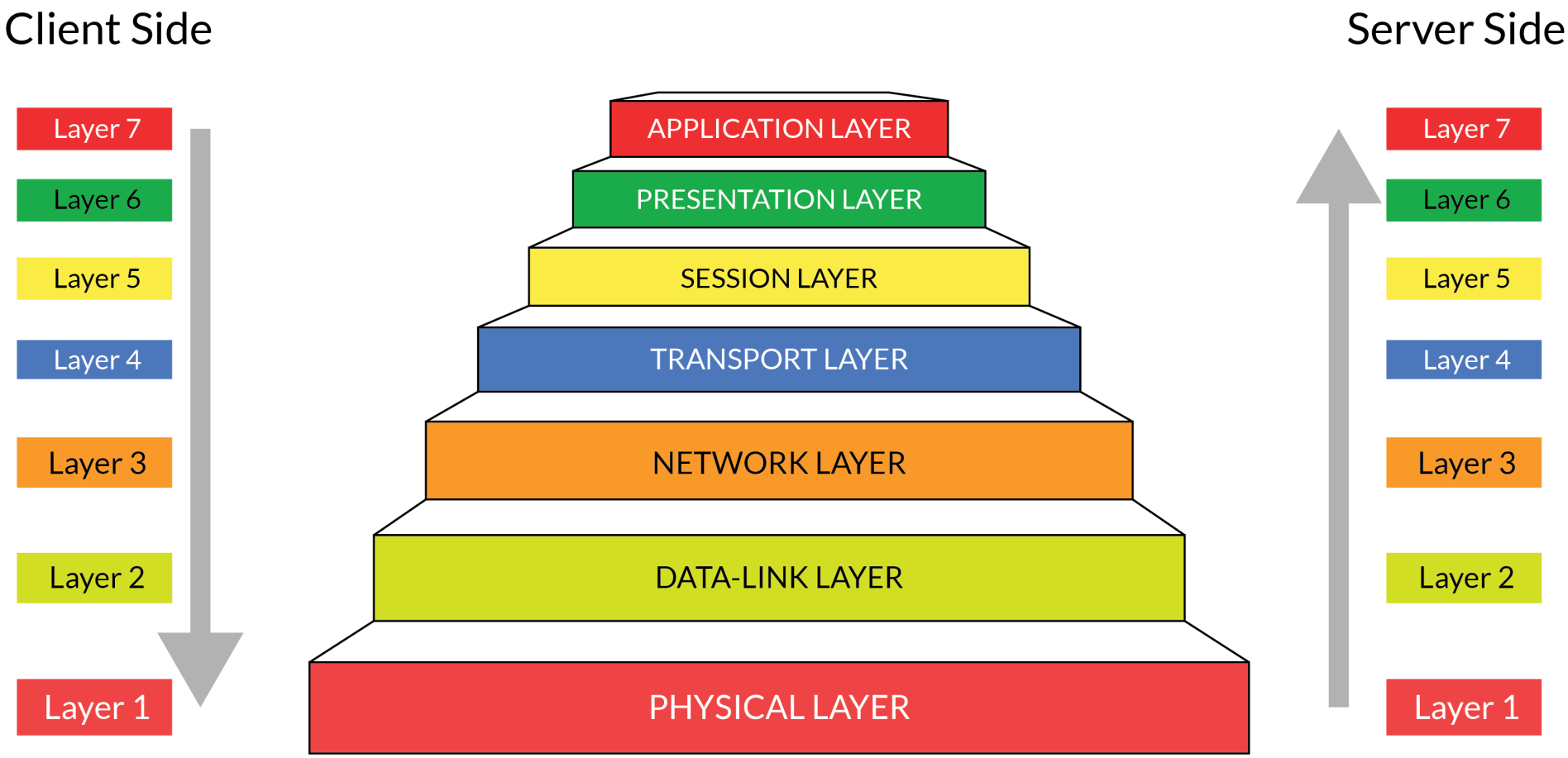
Background

- Networks?
- Distributed computing
- ACID vs. BASE
- Catalogs of Databases

Image generated using Adobe Photoshop.

Background: Networks?

...I thought this was a database class?



Open Systems Interconnection model (OSI model) reference model for how applications communicate over a network. 5

Background: Networked databases

- **Single** database ⇌ **single** client (same host):

  Maybe a network?

- **Single** database ⇌ **single** client (different host):

  Networked

- **Single** database ⇌ **multiple** clients:

  Networked

- **Clustered** database ⇌ **multiple** clients:

  *Networked...maybe **multiple** networks*

Background: Distributed computing

- Terms
- Data replication and consistency
- Consensus algorithms
- CAP Theorem with PACELC Extension
- Types of failures

Image generated using Adobe Photoshop.

Background: Distributed computing

Leveraging *multiple* computers to collaboratively solve tasks over a *network*.

...sounds like the generic case of a clustered database...

Advantages of a clustered solution:

- *Scalable:* Easily add or remove nodes to handle varying loads.
- *Fault Tolerant:* The system remains operational even if some nodes fail.
- *Resource Pool:* Utilize the combined resources of all nodes in the network.

Biggest single challenge: Keeping data **consistent** across nodes can be *difficult*.

Background: Distributed databases key terms

- **Shared-Nothing Architecture**: Each node is independent and self-sufficient, with no shared components that could become a bottleneck.
- **Data Sharding**: The process of splitting a large database into smaller, more manageable pieces called shards, each of which can be stored on a different node.
- **Quorum-Based Replication**: Ensuring data consistency by requiring a majority of nodes (a *quorum*) to agree on the value of data.
- **Two-Phase Commit (2PC)**: A protocol to ensure the consistency of transactions across multiple nodes by using a two-step process.
- **Eventual Consistency**: Updates to the database will propagate to all nodes eventually, but not necessarily immediately, allowing for higher availability.
- **Vector Clocks**: Tracks causal relationships between events and ensure data consistency by associating a timestamp with each update.
- **Scaling**: *Scale-out* (horizontal) → more nodes. *Scale-up* (vertical) → more `resources`.
- **Distributed Transactions**: Spans multiple nodes, requiring coordination.

Background: Data partitioning

Dividing the database into smaller, manageable segments.

Types:

- *Horizontal* partitioning (**sharding**): Distributing rows across multiple nodes.
- *Vertical* partitioning: Distributing columns across different nodes.

Strategies:

- *Range-based:* Assigning data based on a range of values.
- *Hash-based:* Assigning data based on a hash function.
- *List-based:* Assigning data based on a predefined list.
- *Composite:* Combining multiple strategies.

Question:

- *Which type is better?*

Background: Replication

Copying data across different nodes for redundancy and availability.

- *Synchronous* replication: Ensuring data consistency across nodes in real-time.
- *Asynchronous* replication: Allowing for slight delays in data synchronization.

Active/Active:

- Both nodes are active and can handle read and write operations.
- Data is replicated bi-directionally for synchronization.
- Offers high availability and load balancing.

Primary/Replica:

- The primary node handles write operations, while replica nodes are read-only.
- Changes are propagated from the primary to the replicas.
- Provides data redundancy and read scalability.

Background: Data consistency

Ensuring that all nodes in a distributed system have the same data.

Consistency models: Balancing consistency, availability, and partition tolerance.

- *Strong* consistency: Ensuring all nodes have the same data at the same time.
- *Eventual* consistency: Allowing temporary inconsistencies for more performance.
- *Causal* consistency: Orders causally related operations consistently across nodes.
- *Weak* consistency: Delays data updates' visibility to prioritize performance.

Question:

- *Why different models?*

Background: Consensus algorithms

Achieve agreement on a distributed *single data value*, even in the presence of failures.

Paxos:

- Developed by Leslie Lamport in late 80's.
- Theoretical foundation, complex implementation.
- *Active replication:* Each replica executes operations.

Viewstamped Replication (VR):

- Developed by Brian Oki & Barbara Liskov around the same time as Paxos.
- *Passive replication:* Only the primary executes operations.

Raft:

- Developed by Diego Ongaro and John Ousterhout in 2013.
- Designed for simplicity, easier to understand and implement than Paxos.

Background: Raft example

https://raft.github.io/

Background: Types of Failures in Distributed Systems

Network Failures: Communication issues between nodes: soft (TTL) / hard (partitions).

Node Failures: Hardware/software failures that cause a node to become unresponsive.

Replication Failures:

- **Bad Data Replication:** Incorrect/corrupted data is replicated across nodes.
- **Replication Lag:** Delays in replicating data from the primary to replicas.
- **Conflicts:** Simultaneous updates to the same data can lead to value conflicts.

Consistency Failures: Violations of the consistency model → incorrect data being read.

Byzantine Failures: Malicious or arbitrary failures where a node behaves unpredictably.

Question:

- *What is "blast radius"?*

Background: Failure effect "Split-Brain"

- A split-brain occurs when a network partition divides a cluster into subclusters.
- Each subcluster may operate independently, risking data inconsistency.
- Solutions include quorum-based decisions, fencing, and manual intervention.

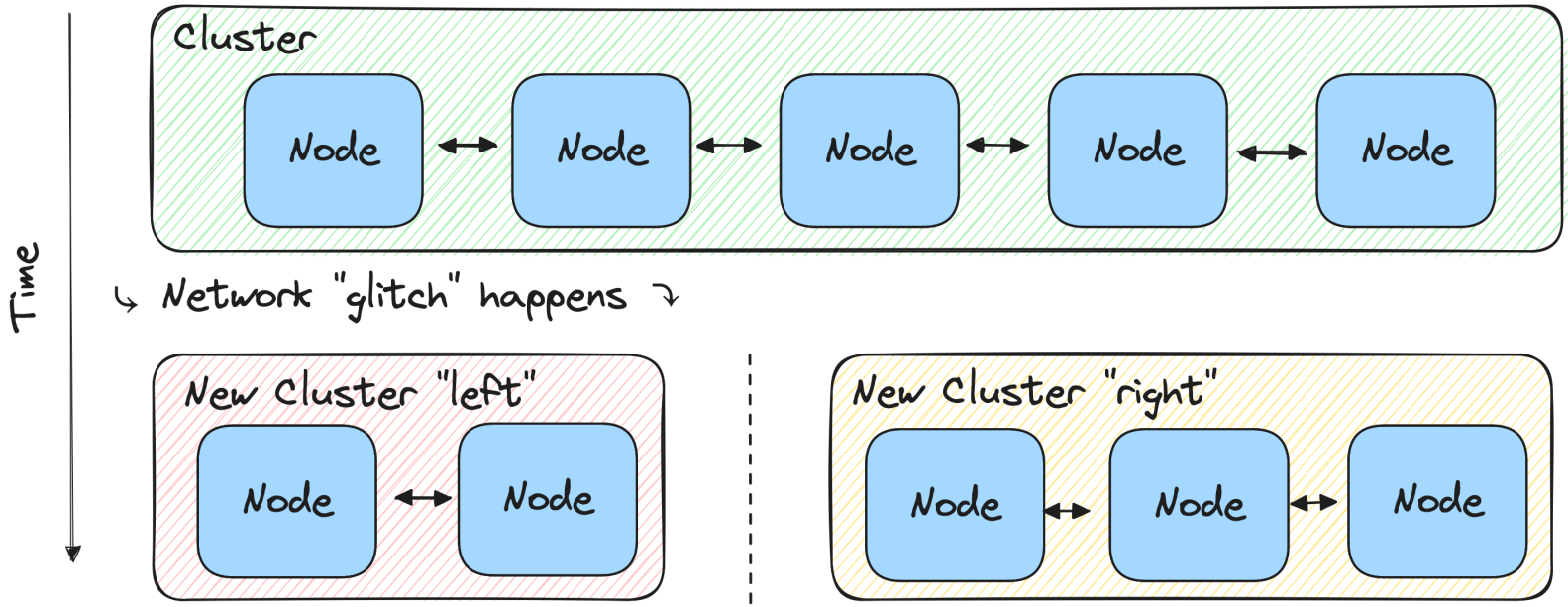
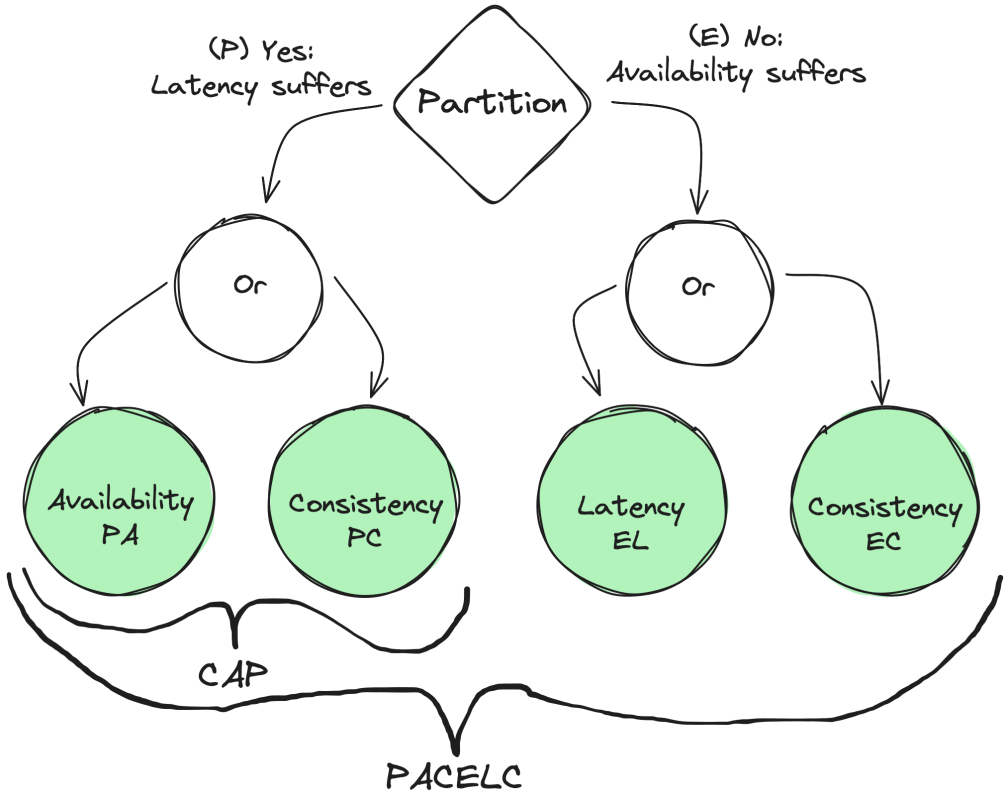


Illustration of a split-brain scenario caused by a network partition in a clustered system, leading to two separate subclusters operating independently.

Background: CAP Theorem with PACELC Extension

CAP Theorem (2000): any distributed data store can provide only 2 of 3 guarantees.
PACELC Extension (2012): extends CAP for partitioned and non-partitioned states.



If a partition (P) occurs, the system must balance availability (A) and consistency (C); otherwise (E), in normal operation without partitions, it must balance latency (L) and consistency (C).

Background: Back to Databases with 'ACID' vs. 'BASE'

ACID:

- ***A*tomicity:** Transactions are all-or-nothing.
- ***C*onsistency:** Database stays consistent before/after transactions.
- ***I*solation:** Transactions are executed independently.
- ***D*urability:** Committed transactions are permanent.

ACID is used in relational databases for strict consistency.

BASE:

- ***B*asically *A*vailable:** Data is available, but not always up-to-date.
- ***S*oft state:** System state may change without input.
- ***E*ventual Consistency:** Consistency achieved over time, not immediately.

BASE is common in distributed databases for scalability and availability.

Background: Back to networks...

The "Fallacies of distributed computing" are:

```
1. The network is reliable;
2. Latency is zero;
3. Bandwidth is infinite;
4. The network is secure;
5. Topology doesn't change;
6. There is one administrator;
7. Transport cost is zero;
8. The network is homogeneous.
```

Sun Microsystems' Peter Deutsch (1994).

Questions:

- *What are the implications for networked databases?*
- *What are the implications for distributed databases?*
- *Better to scale "up" or scale "out"?*

Background: Database of Databases

The On-line Database of Databases: https://dbdb.io/

- Created and maintained by the *Carnegie Mellon University Database Group*
- Provides detailed information about a wide range of database systems, including their features, history, and underlying technologies.
- Contains some very esoteric (academic/specialized) databases.

DB-engines: https://db-engines.com/

- Created and maintained by an Austrian IT consulting company.
- Provides "ranking" (popularity method) of databases with metadata about them.
- Contains more "mainstream" databases.
- Encyclopedia Index is a glossary of database terms.

Physalia

- Overview
- Handling failures
- NSDI '20 presentation

Portuguese man-of-war (Physalia physalis). Image courtesy of NSDI '20 AWS Physalia presentation.

Physalia: An Overview

"Millions of tiny databases" by Marc Brooker, Tao Chen, and Fan Ping. DOI

NSDI'20: Proceedings of the 17th Usenix Conference on Networked Systems Design and Implementation, February 2020, Pages 463–478

- Stores configuration info for the AWS Elastic Block Storage (EBS) control plane.
- Strongly consistent, critical for replication protocol correctness.
- Consists of numerous small databases (cells) for fault isolation

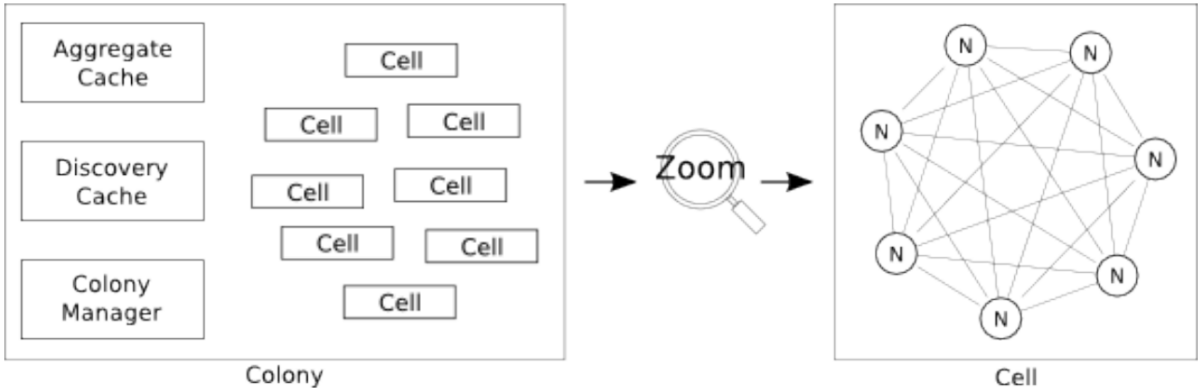


Figure 2: Overview of the relationship between the colony, cell and node.

Physalia: Handling failures

- Designed to minimize the blast radius of network partitions and fail gracefully.
- Uses chain-replication and Paxos-based consensus for reconfiguration.
- Incrementally deployemented to limit blast radius, with careful color-coded zones.
- Rigorous testing, including TLA$^+$ protocols modeling and Jepsen tests for linearizability under network failures.

For a more in-depth exploration, read Adrian Colyer's summary on The Morning Paper.

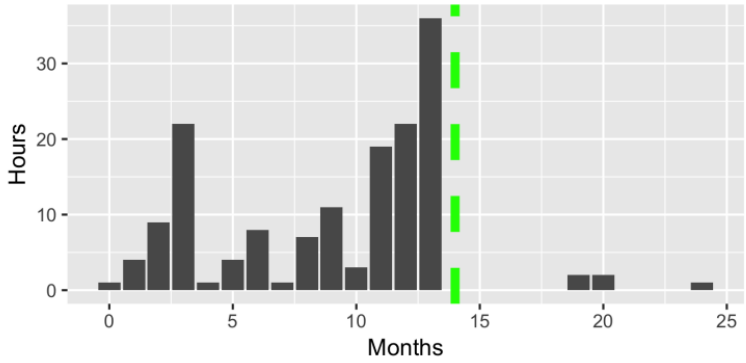
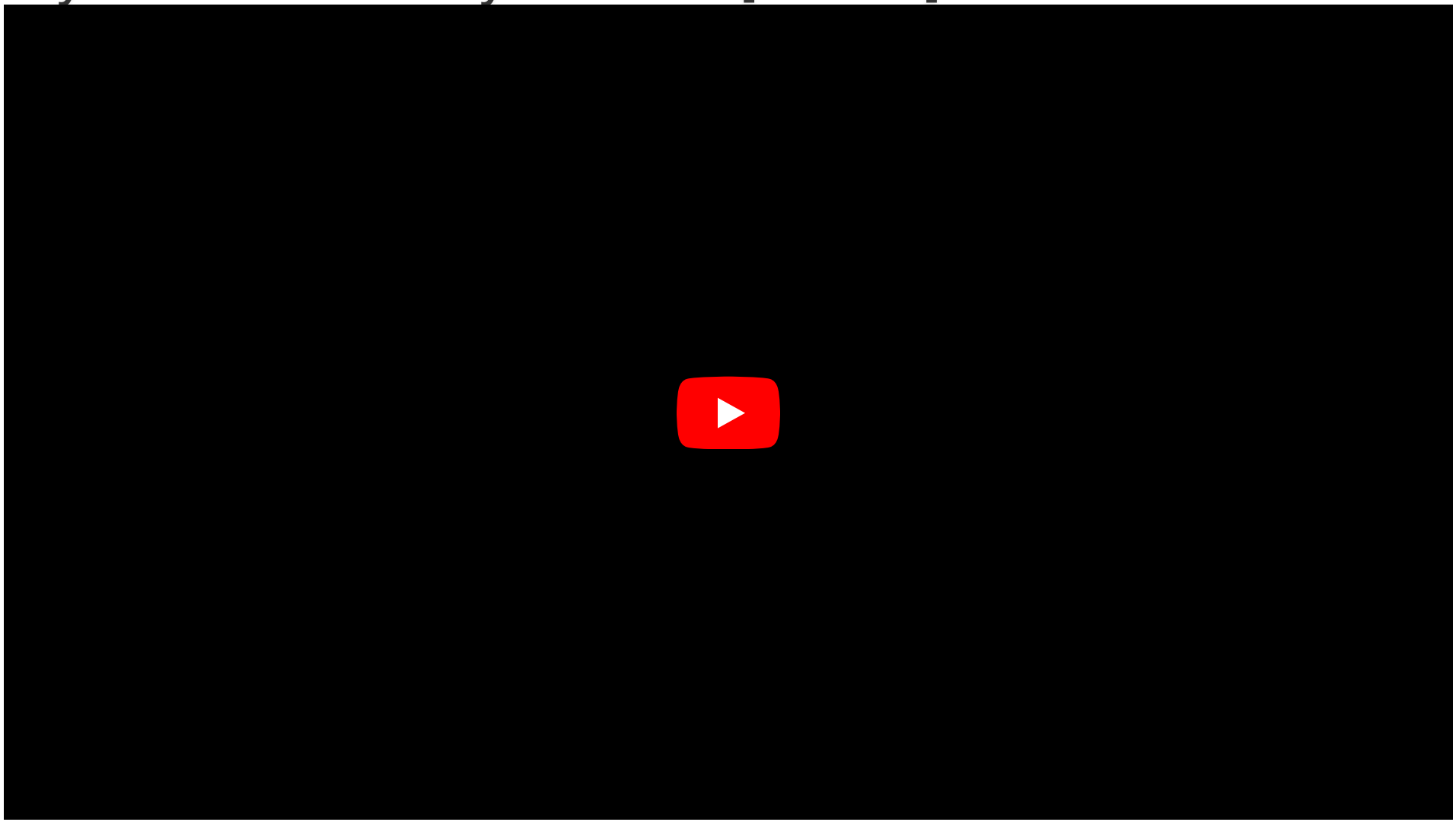


Figure 9: Number of hours per month where EBS masters experienced an error rate > 0.05% in a production colony. The vertical line shows the deployment of Physalia.

Physalia: Millions of Tiny Databases [NSDI '20]



NSDI '20 presentation

CockroachDB

- Overview
- Transaction techniques
- SIGMOD '20 presentation

Cockroach Labs logo. Image courtesy of Cockroach Labs.

CockroachDB: An Overview

"CockroachDB: The Resilient Geo-Distributed SQL Database" by Cockroach Labs, Inc. DOI

SIGMOD '20: Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data, June 2020, Pages 1493–1509

- ACID-compliant and provides transactional guarantees.
- Fault tolerant and highly availability, geo-distributed partitioned and replica placed, with high-performance transactions.

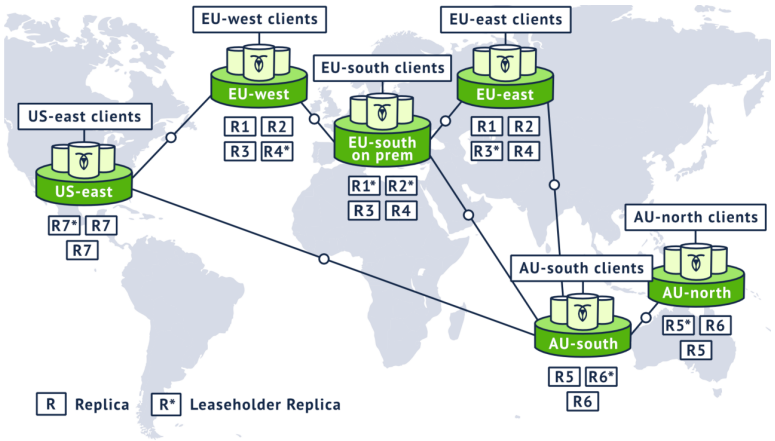


Figure 1: A global CockroachDB cluster (showing Replicas and Leaseholder Replicas)

CockroachDB: Transactions techniques

- Reduces commit wait times with transactional *Write Pipelining*.
- Speeds up transaction completion with *Parallel Commits*.
- Ensures data consistency and durability through *Raft* consensus.
- Indicates pending writes for isolation with *Write Intents*.
- Helps maintain transaction order through a *Timestamp Cache*.

(Also wire-compatible with PostgreSQL, allowing for easy migration.)

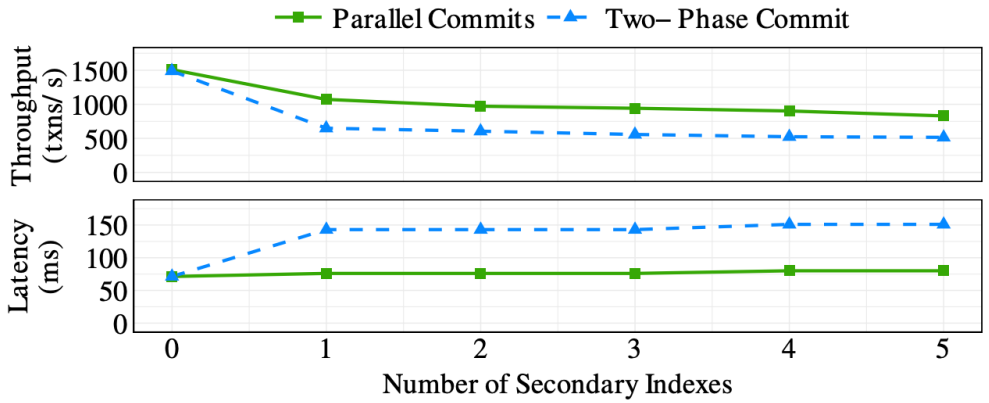


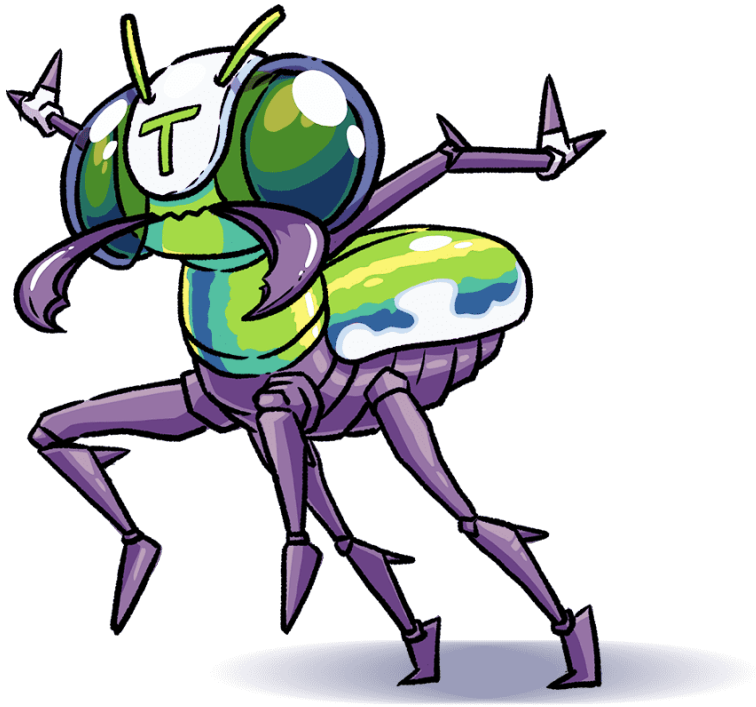
Figure 2: Performance impact of Parallel Commits

CRDB: The Resilient Geo-Distributed SQL Database [SIGMOD '20]

SIGMOD '20 presentation

TigerBeetle

- Overview
- Safety and testing
- Simulator presentation

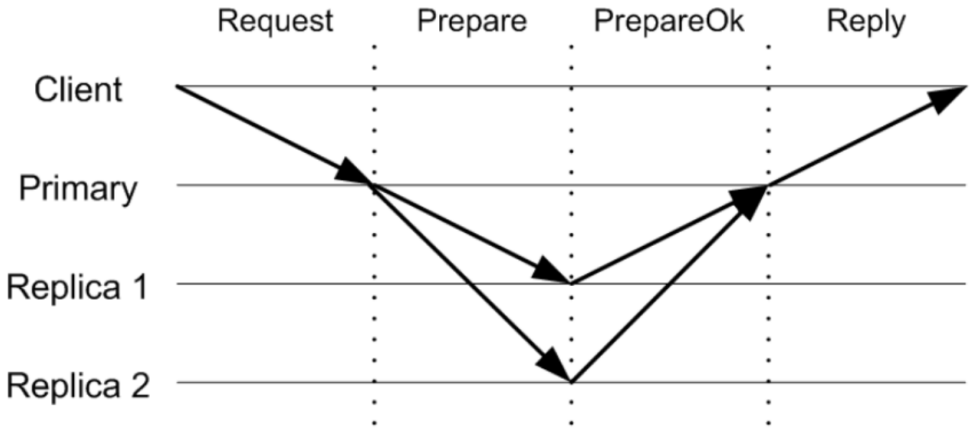


TigerBeetle mascot. Courtesy of TigerBeetle Inc.

TigerBeetle: An Overview

TigerBeetle: A High-Performance Financial Accounting Database by TigerBeetle Inc.

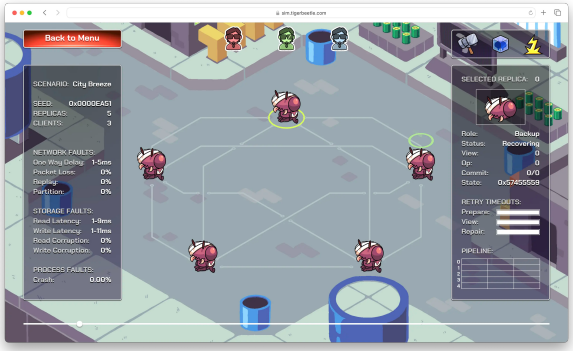
- Purpose-built for financial transactions to ensure critical safety and performance.
- Adheres to double-entry accounting principles for financial consistency.
- ACID-compliant with two-phase transfers for reliable staged fund movements.
- Aims for 1M transfers per second on OTS hardware.
- Viewstamped Replication & Protocol-Aware Recovery for Consensus-Based Storage.



Viewstamped Replication: Normal case processing in VR for a configuration with $f = 1$

TigerBeetle: Safety and testing

- Utilizes the Zig programming language for safety features such as OOM safety, bounds checking, and explicit control flow.
- Protects against varying storage-related failures with a robust storage fault model.
- Rigorous testing regime, including automated testing, audits of consensus and replication protocols, and state machine verification.
- *Viewstamped Operation Replicator (VOPR)* is a deterministic game to simulate TigerBeetle's protocol to test distributed behaviors.



VOPR Level 1: City Breeze, simulating a perfect environment. No network partitions. No packet loss or replays. No crashes o disk corruption, and, of course, low latency network and disk I/O.

TigerBeetle: SimTigerBeetle [The TigerBeetle Sessions]

SimTigerBeetle Director's Cut!: SimTigerBeetle, a TigerBeetle simulator in your browser.

Conclusion

- **Evolution of Databases:** From primitive to NewSQL and distributed databases, adapting to modern challenges like scalability, availability, and security.
- **Distributed Systems:** Key concepts like CAP/PACELC, consensus algorithms, and types of failures highlight the complexities of distributed computing.
- **Physalia:** Demonstrates fault isolation and graceful failure handling in AWS' EBS contro plane.
- **CockroachDB:** Combines ACID compliance with fault tolerance and high availability in a geo-distributed setup.
- **TigerBeetle:** Specializes in financial transactions with high performance and safety, leveraging Zig for robustness.

Thank you

Dr. Jacob Hochstetler
Distinguished Engineer, Vice President, Fidelity Investments
Clinical Assistant Professor, University of North Texas
Jacob.Hochstetler@UNT.edu
Jacob.Hochstetler@Fidelity.com
https://github.com/jh125486