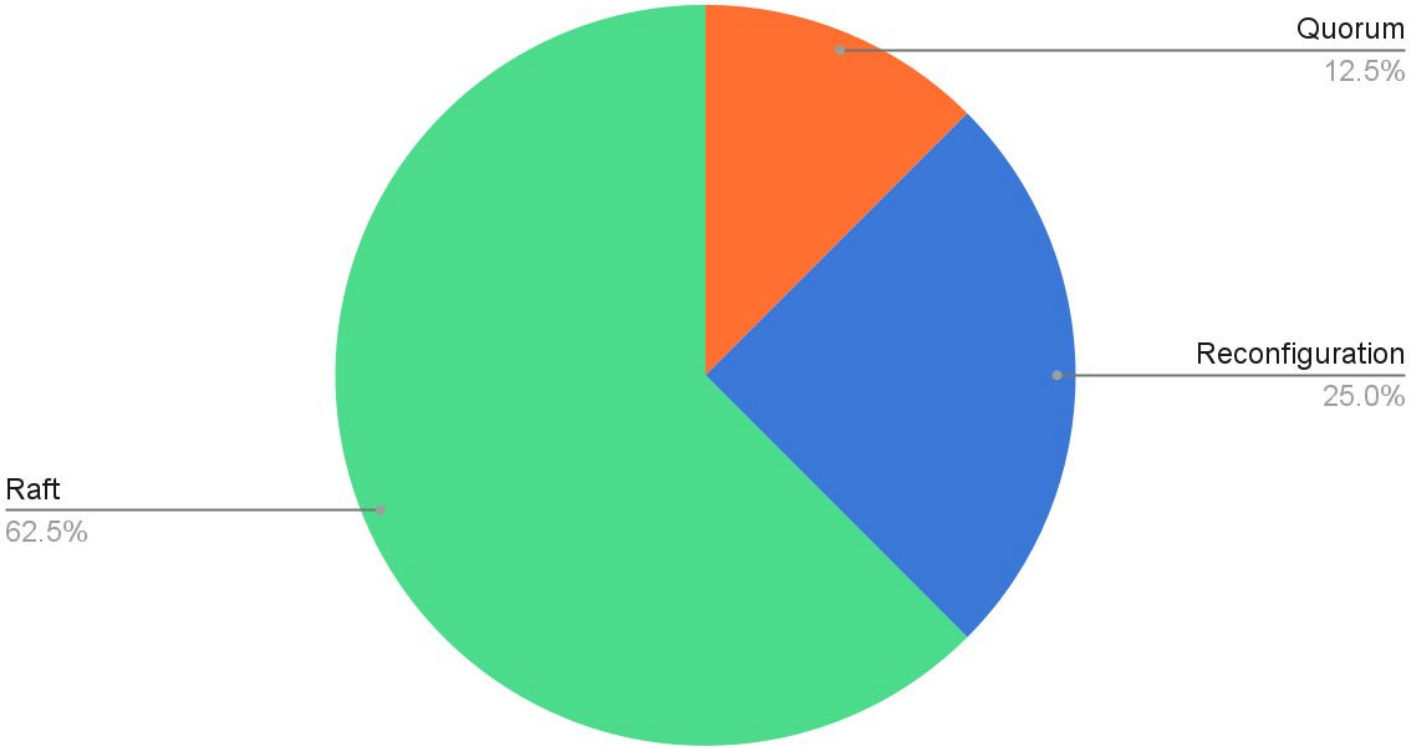
Enough With All The Raft

Databases by Replication Algorithm



Y **Hacker News** new | threads | past | comments | ask |

▲ Ask HN: Should I use Raft, Multi-Paxos, or VSR?

255 points by everyone 1 year ago | hide | past | favorite | 1000 comments

Replication Algorithms

1) Quorums

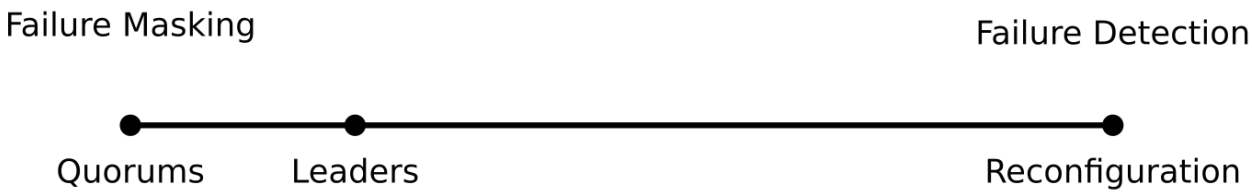
- Majority Quorums
- Paxos

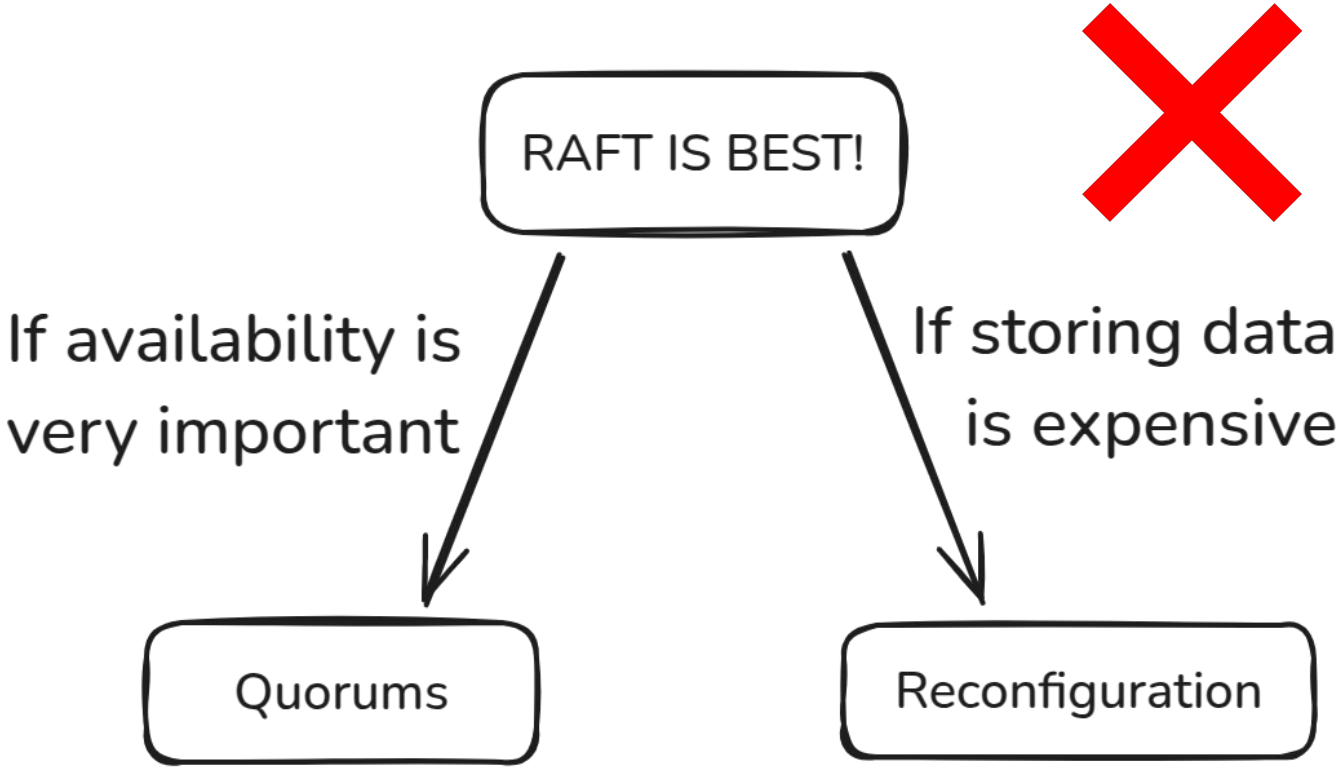
2) Leaders

- Raft
- Multi-Paxos
- VSR

3) Reconfiguration

- Primary-Backup
- Chain Replication




RAFT IS BEST!
If availability is very important
If storing data is expensive
Quorums
Reconfiguration

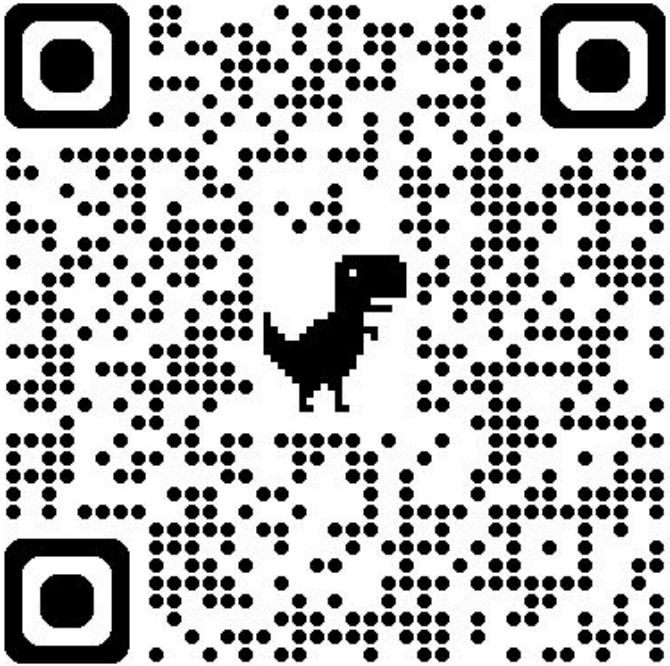
"Raft is the Best!"

It’s best at… what?

Raft is Best!™

It’s best at…

- Latency?
- Throughput?
- Storage space?



https://transactional.blog/blog/2024-data-replication-design-spectrum

	Replicas Required for f=2	Storage Efficiency	Read Bandwidth Efficiency	Write Bandwidth Efficiency	Chance of Unavailability on Failure	Read Latency	Write Latency
Paxos (Quorum)	5	20%	20%	20%	0%	1RTT	2RTT
PacificA (Reconfig)	3	33%	100%	16.7%	100%	1-2RTT	2RTT
Follower Reads (Raft)	5	20%	100%	5%	20%	1-2RTT	2RTT

It’s not though…

"Raft is better because Raft is Simple!"

It’s simpler than Multi-Paxos

On Complexity

1) Quorums

Hard: Large state space to test

Easy: Slow == Failed

2) Leaders

Hard: Large state space to test & Leader liveness

Easy: N/A

3) Reconfiguration

Hard: Replica liveness

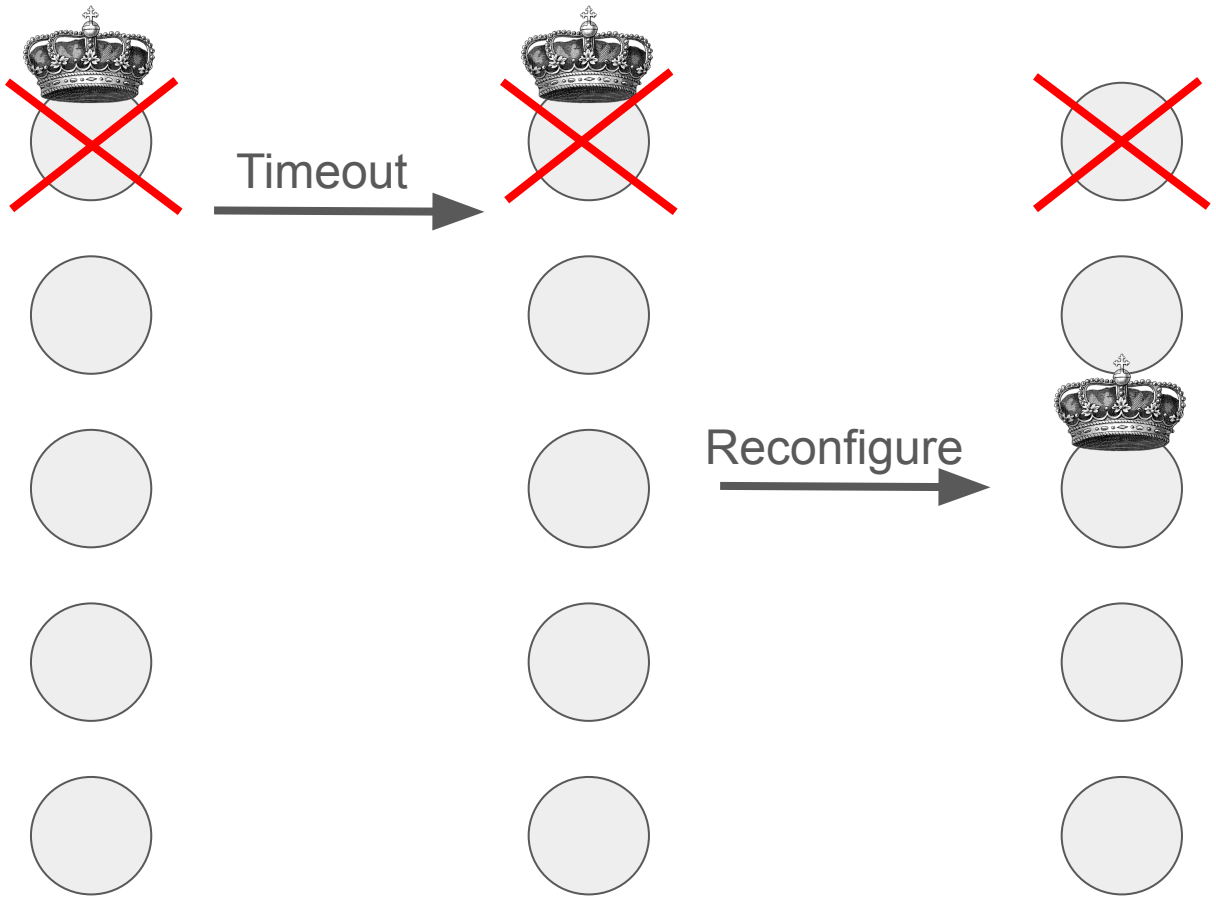
Easy: No error handling

"Raft is better than Reconfiguration because Reconfiguration has unavailability!"

No.

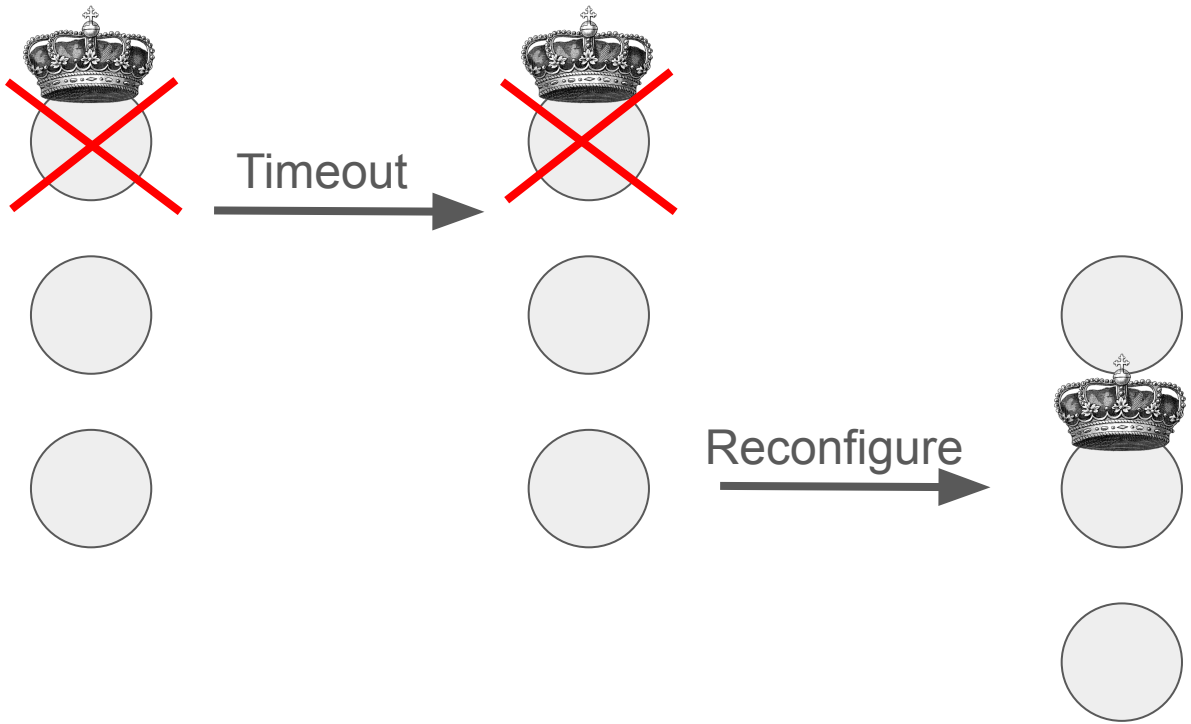
1.

Raft

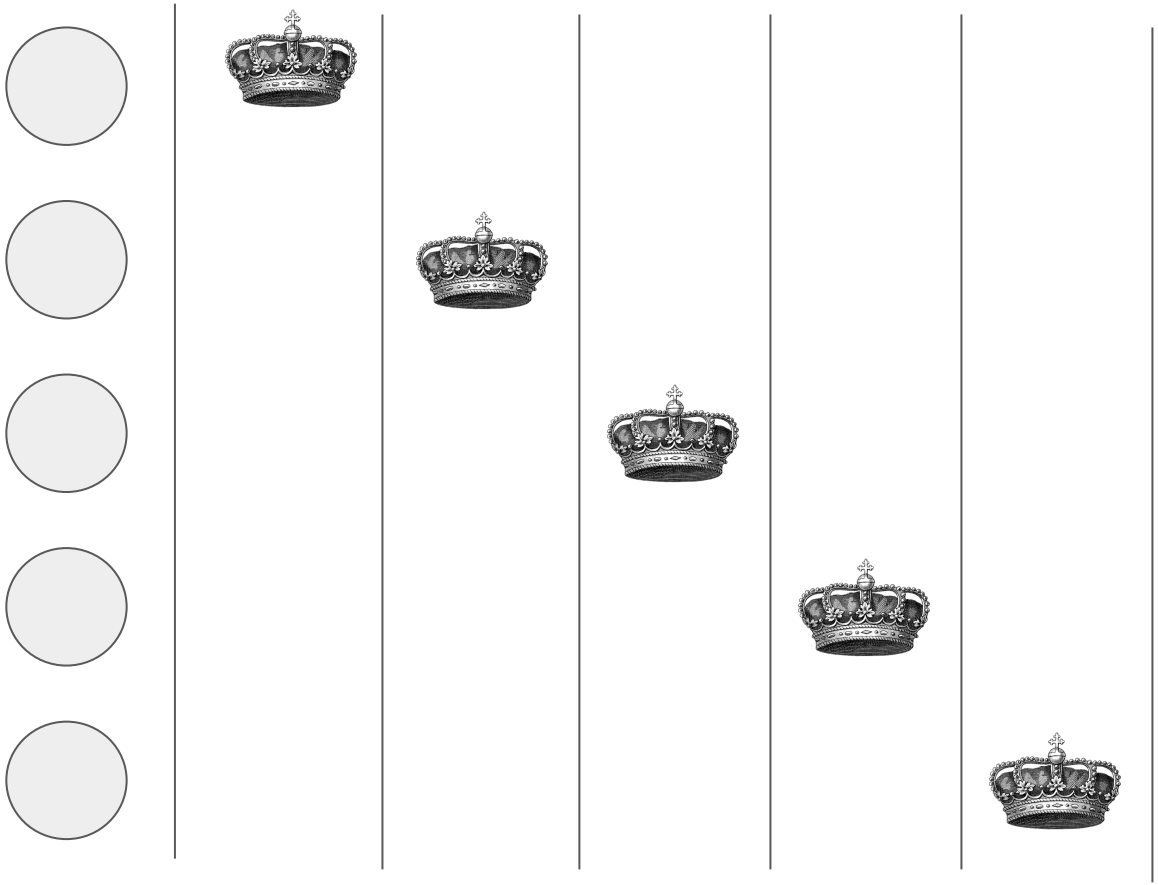


1.

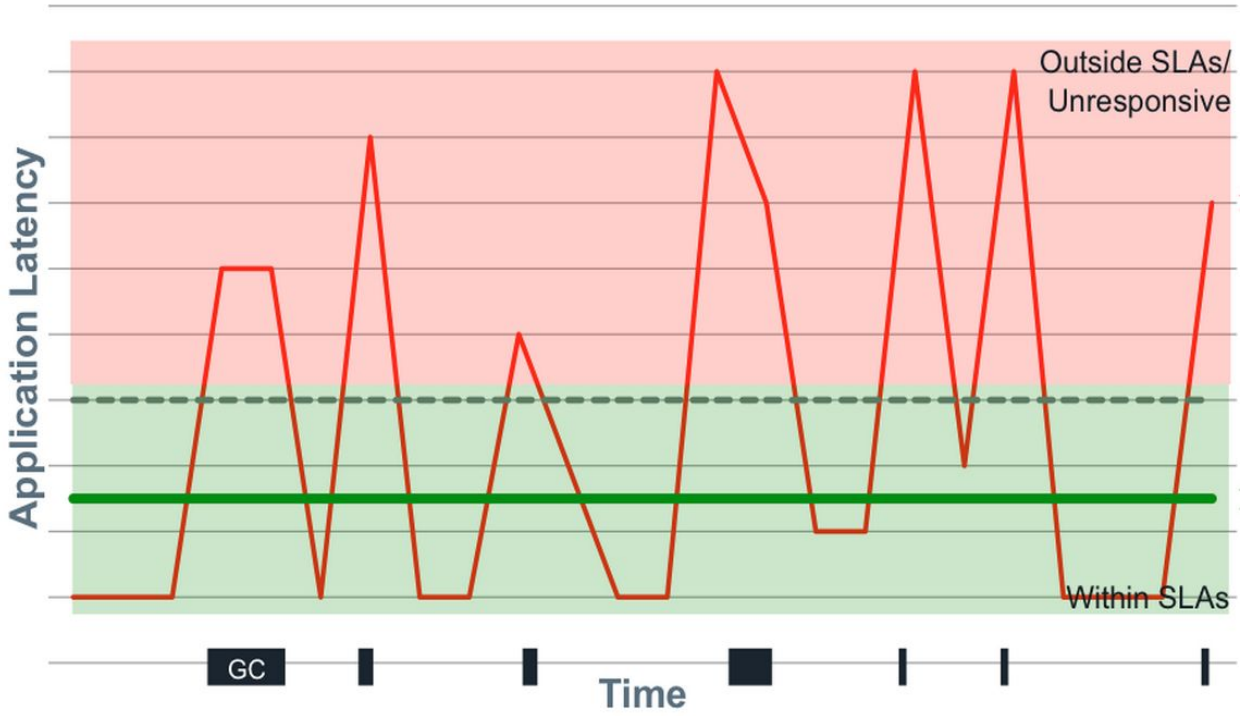
Primary-Backup



2. It’s always unavailable for someone…



3.



Credit: https://bravenewgeek.com/everything-you-know-about-latency-is-wrong/

3.

https://arxiv.org/abs/2412.02792

Taurus Database: How to be Fast, Available, and Frugal in the Cloud

Alex Depoutovitch, Chong Chen, Jin Chen, Paul Larson, Shu Lin, Jack Ng, Wenlin Cui, Qiang Liu, Wei Huang, Yong Xiao, Yongjun He
Huawei Research Canada

ABSTRACT

Using cloud Database as a Service (DBaaS) offerings instead of on-premise deployments is increasingly common. Key advantages include improved availability and scalability at a lower cost than on-premise alternatives. In this paper, we describe the design of Taurus, a new multi-tenant cloud database system. Taurus separates the compute and storage layers in a similar manner to Amazon Aurora and Microsoft Socrates and provides similar benefits, such as read replica support, low network utilization, hardware sharing and scalability. However, the Taurus architecture has several unique advantages. Taurus offers novel replication and recovery algorithms providing better availability than existing approaches using the same or fewer replicas. Also, Taurus is highly optimized for performance, using no more than one network hop on critical paths and exclusively using append-only storage, delivering faster writes, reduced device wear, and constant-time snapshots. This paper describes Taurus and provides a detailed description and analysis of the storage node architecture, which has not been previously available from the published literature.

in the Cloud. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data (SIGMOD'20), June 14–19, 2020, Portland, OR, USA.* ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3318464.3386129

1 INTRODUCTION

As companies move their applications to the cloud, demand for cloud-based relational database service (DBaaS) is growing rapidly. Amazon, Microsoft, Alibaba, and other cloud providers all offer such services. Most DBaaS offerings were initially based on traditional monolithic database software, essentially just running databases on (virtual) machines in the cloud, using either local storage or cloud storage. Although simple to implement, this approach cannot provide what customers want from a cloud database service [12]. From a customer's point of view, an ideal database service should be highly available, require no maintenance, and scale up and down automatically with database size and workload. It should also deliver high performance, be low cost, and users should pay only for resources actually used (pay-as you-go). These goals can't be achieved by running the same

3.

Replication method	Probability of non-availability		x = 0.15		x = 0.05		x = 0.01	
	Write	Read	Write	Read	Write	Read	Write	Read
$N = 6, N_W = 4, N_R = 3$	$20 * x^3$	$15 * x^4$	$7 * 10^{-2}$	$8 * 10^{-3}$	$3 * 10^{-3}$	10^{-4}	$2 * 10^{-5}$	$2 * 10^{-7}$
$N = 3, N_W = 2, N_R = 2$	$3 * x^2$	$3 * x^2$	$7 * 10^{-2}$	$7 * 10^{-2}$	$8 * 10^{-8}$	$8 * 10^{-3}$	$3 * 10^{-4}$	$3 * 10^{-4}$
$N = 3, N_W = 3, N_R = 1$	$3 * x$	x^3	$5 * 10^{-1}$	$3 * 10^{-3}$	$2 * 10^{-1}$	10^{-4}	$3 * 10^{-2}$	10^{-6}
Taurus	0	x^3	0	$3 * 10^{-3}$	0	10^{-4}	0	10^{-6}

Table 1: Comparing the probability of the storage being unavailable for Taurus and common quorum replication variants

> Unlike pure quorum writes, Taurus log writes don't need to land on specific Log Store nodes, so formula 1 is not applicable. The probability of the storage layer being unavailable for writes due to independent node failures is close to zero for a cluster of hundreds of nodes because if a chosen node is unavailable, any other node can be chosen instead. Individual node failures affect latency, as failed writes have to be retried with a different set of Log Store nodes, but they don't affect availability.

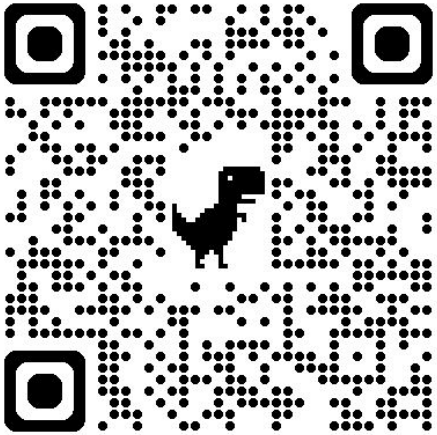
"Raft is better than Reconfiguration because Reconfiguration needs a consensus service!"

Yes: Kubernetes Coordination API,
S3, Dynamo, Postgres, etc.

PacificA: Replication in Log-Based Distributed Storage Systems

Wei Lin, Mao Yang
Microsoft Research Asia
{weilin, maoyang}@microsoft.com

Lintao Zhang, Lidong Zhou
Microsoft Research Silicon Valley
{lintaoz, lidongz}@microsoft.com



ABSTRACT

Large-scale distributed storage systems have gained popularity for storing and processing ever increasing amount of data. Replication mechanisms are often key to achieving high availability and high throughput in such systems. Research on fundamental problems such as consensus has laid out a solid foundation for replication protocols. Yet, both the architectural design and engineering issues of practical replication mechanisms remain an art.

ple instances that co-exist.

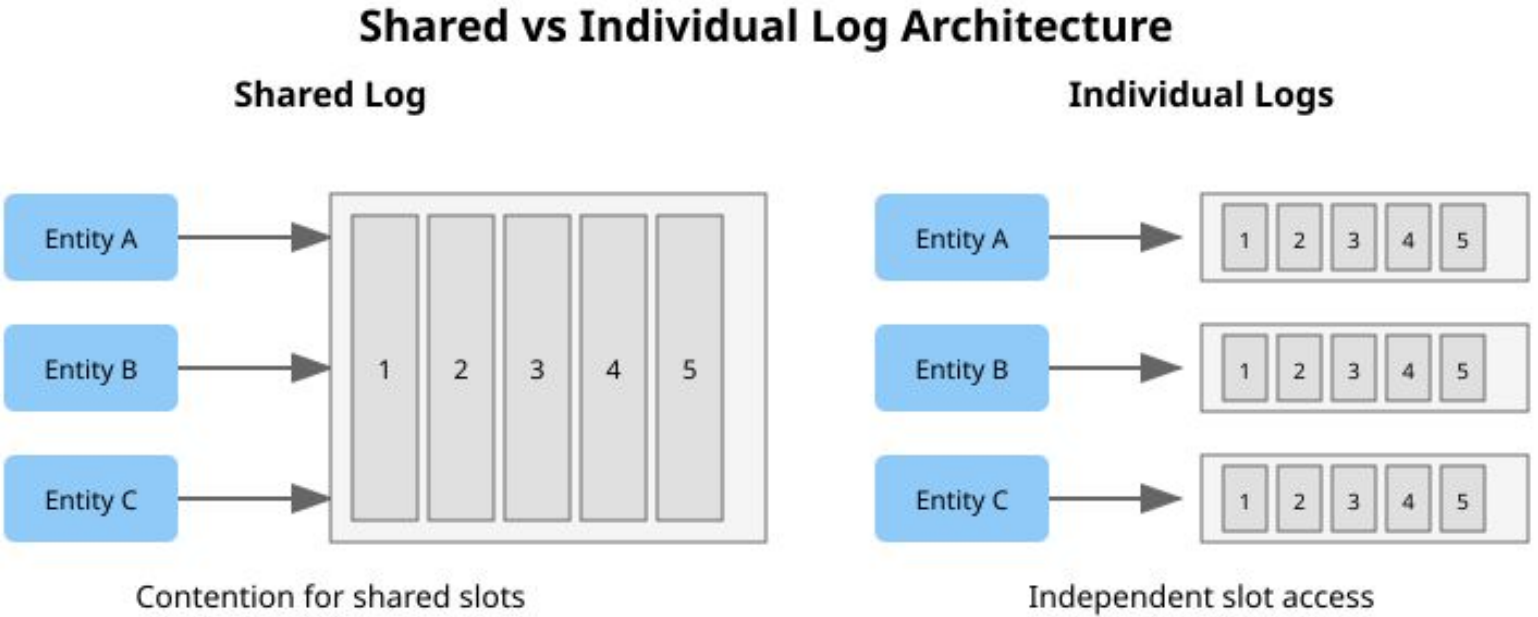
In this paper, we describe our experience in designing, implementing, and evaluating the replication mechanism for large-scale log-based storage system in a local-area-network (LAN) cluster environment. Such log-based designs are commonly used in storage systems (e.g., [12, 13, 15, 24]) to improve performance by transforming random writes to sequential writes and by allowing batching, as well as to support transactional semantics. Many recently proposed stor-

https://www.microsoft.com/en-us/research/wp-content/uploads/2008/02/tr-2008-25.pdf

"Raft is better than Quorums because Quorums livelock on contention!"

For consensus, mostly.

1. Stop serializing things which don’t need to be serialized



1.

CASPaxos: Replicated State Machines without logs

Denis Rystsov
Microsoft
derystso@microsoft.com

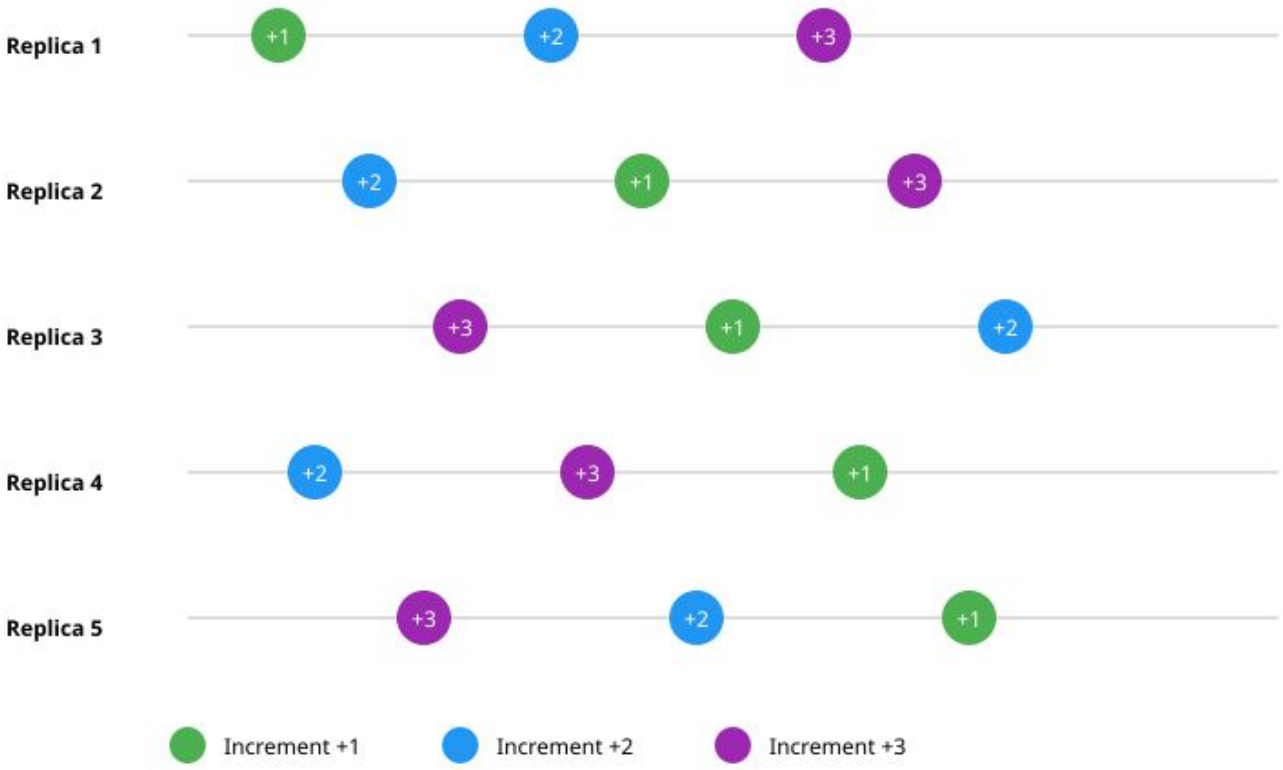
Abstract

CASPaxos is a wait-free, linearizable, multi-writer multi-reader register in unreliable, asynchronous networks supporting arbitrary update operations including compare-and-set (CAS). The register acts as a replicated state machine providing an interface for changing its value by applying an arbitrary user-provided function (a command). Unlike Multi-Paxos and Raft which replicate the log of commands, CASPaxos replicates state, thus avoiding associated complexity, reducing write amplification, increasing concurrency of disk operations and hardware utilization.

The paper describes CASPaxos, proves its safety properties and evaluates the characteristics of a CASPaxos-based prototype of key-value storage.

https://arxiv.org/abs/1802.07000

2. Stop serializing commutative operations



2.

The Escrow Transactional Method

PATRICK E. O'NEIL
Computer Corporation of America

A method is presented for permitting record updates by long-lived transactions without forbidding simultaneous access by other users to records modified. Earlier methods presented separately by Gawlick and Reuter are comparable but concentrate on "hot-spot" situations, where even short transactions cannot lock frequently accessed fields without causing bottlenecks. The Escrow Method offered here is designed to support nonblocking record updates by transactions that are "long lived" and thus require long periods to complete. Recoverability of intermediate results prior to commit thus becomes a design goal, so that updates as of a given time can be guaranteed against memory or media failure while still retaining the prerogative to abort. This guarantee basically completes phase one of a two-phase commit, and several advantages result: (1) As with Gawlick's and Reuter's methods, high-concurrency items in the database will not act as a bottleneck; (2) transaction commit of different updates can be performed asynchronously, allowing natural distributed transactions; indeed, distributed transactions in the presence of delayed messages or occasional line disconnection become feasible in a way that we argue will tie up minimal resources for the purpose intended; and (3) it becomes natural to allow for human interaction in the middle of a transaction without loss of concurrent access or any special difficulty for the application programmer. The Escrow Method, like Gawlick's Fast Path and Reuter's Method, requires the database system to be an "expert" about the type of transactional updates performed, most commonly updates involving incremental changes to aggregate quantities. However, the Escrow Method is extendable to other types of updates.

https://mwhittaker.github.io/papers/html/o1986escrow.html

3.

CEP-15: Fast General Purpose Transactions

Elliott Smith, Benedict
benedict@apple.com

Zhang, Tony
nudzhang@umich.edu

Eggleston, Blake
beggleston@apple.com

Andreas, Scott
cscotta@apple.com

Abstract

Modern applications replicate and shard their state to achieve fault tolerance and scalable performance. This presents a coordination problem that modern databases address using leader-based techniques that entail trade-offs: either a scalability bottleneck or weaker isolation. Recent advances in leaderless protocols that claim to address this coordination problem have not yet translated into production systems. This paper outlines distinct performance compromises entailed by existing leaderless protocols in comparison to leader-based approaches. We propose techniques to address these short-comings and describe a new distributed transaction protocol ACCORD, integrating these techniques. ACCORD is the first protocol to achieve the same steady-state performance as leader-based protocols under important conditions such as contention and failure, while delivering the benefits of leaderless approaches to scaling, transaction isolation and geo-distributed client latency. We propose that this combination of features makes ACCORD uniquely suitable for implementing general purpose transactions in Apache Cassandra.

https://cwiki.apache.org/confluence/display/CASSANDRA/CEP-15%3A+General+Purpose+Transactions

1) Quorums

Best For:

- Low Tail Latency

Fatal Flaw:

- High Contention Livelock

2) Leaders

Best For:

- None

Fatal Flaw:

- None

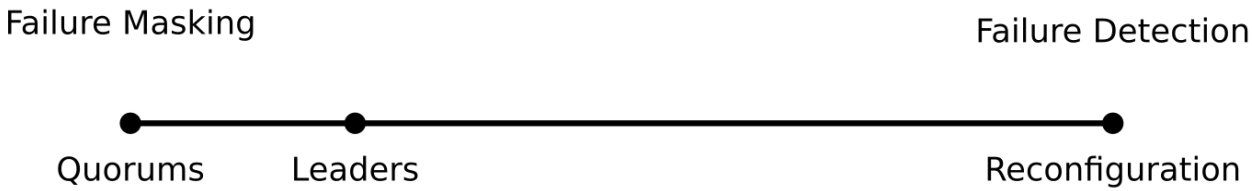
3) Reconfiguration

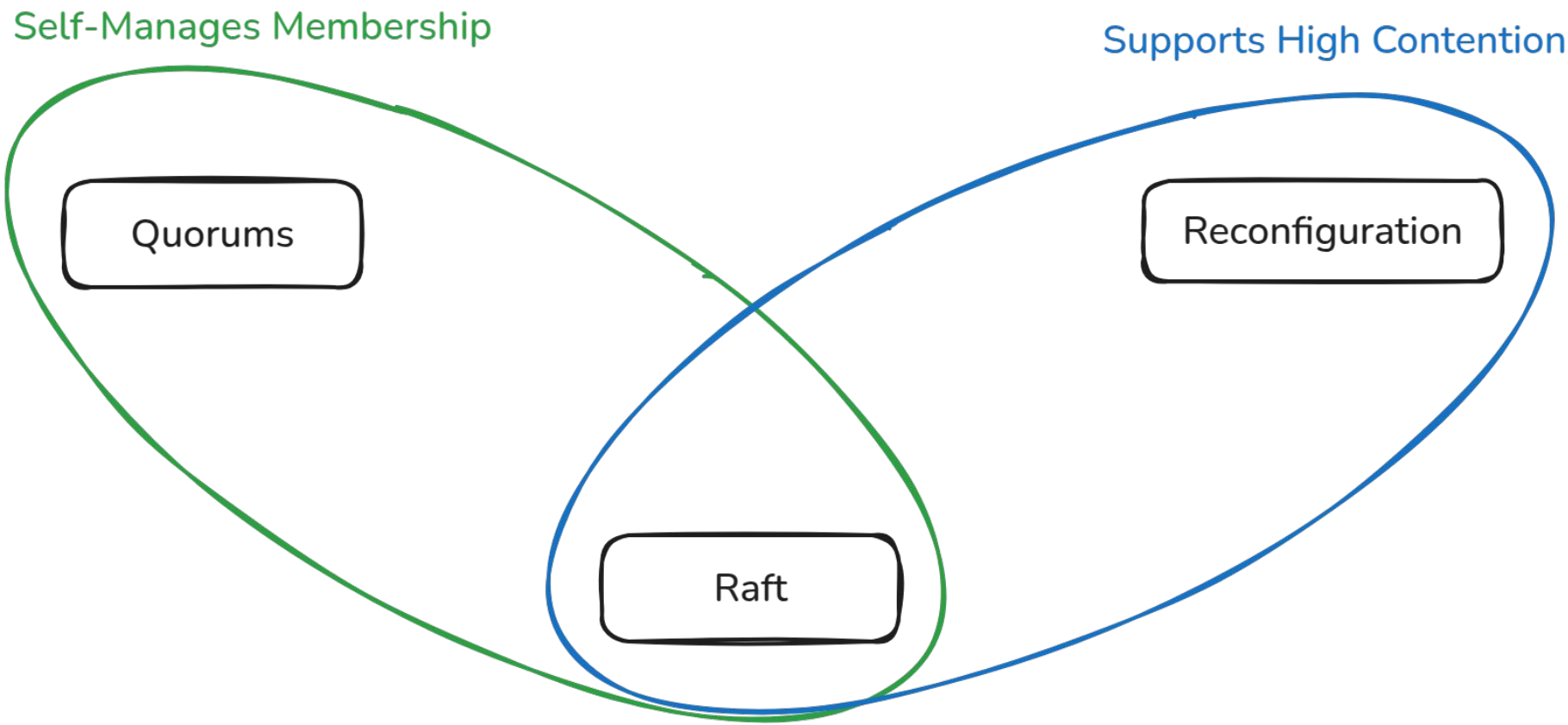
Best For:

- Cost Efficiency

Fatal Flaw:

- External Membership Service

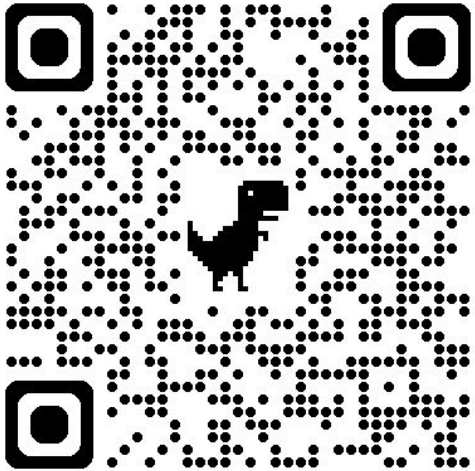



Self-Manages Membership
Supports High Contention
Quorums
Reconfiguration
Raft

Use the replication algorithm that best fits your use case

Replication Comparison Reading: Disaggregated OLTP

System	Replication Type
Aurora	Quorums (With elected writer)
Socrates	Blob Storage
PolarDB (PolarFS)	Raft
Taurus	Reconfiguration



https://transactional.blog/notes-on/disaggregated-oltp