# Brief Announcement: Racos: a Leaderless Erasure Coding State Machine Replication

Jonathan Zarnstorff*
jonathanzarnstorff@gmail.com
Unaffiliated
Boston, MA, USA

Lucas Lebow
lucasphone75@gmail.com
Unaffiliated
Denver, CO, USA

Dillon Remuck
dremuck@clarku.edu
Clark University
Worcester, MA, USA

Colin Ruiz
cruiz@clarku.edu
Clark University
Worcester, MA, USA

Lewis Tseng*
lewistseng@acm.org
Clark University
Worcester, MA, USA

## ABSTRACT

Cloud storage systems often use state machine replication (SMR) to ensure reliability and availability. Erasure coding has recently been integrated with SMR to reduce disk and network I/O costs. This brief announcement shares our experience in developing a *leaderless* erasure coding SMR system. We integrate our system Racos with etcd, a distributed key-value storage that powers Kubernetes. Racos outperforms competitors by up to 3.36x in throughput.

## CCS CONCEPTS

• **Computer systems organization → Reliability**.

## KEYWORDS

State machine replication, Erasure coding, Leaderless, Rabia

**ACM Reference Format:**
Jonathan Zarnstorff*, Lucas Lebow, Dillon Remuck, Colin Ruiz, and Lewis Tseng. 2024. Brief Announcement: Racos: a Leaderless Erasure Coding State Machine Replication. In *Proceedings of the 36th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA '24), June 17–21, 2024, Nantes, France.* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3626183.3660273

## 1 INTRODUCTION

Modern distributed systems in clouds often use state machine replication (SMR) [15] to guarantee reliability and availability. SMR provides strong consistency, which simplifies and lowers the development and deployment efforts [4, 5, 7]. Production systems usually use Multi-Paxos [10] or Raft [12] to implement the SMR component, e.g., Google Spanner [7], Microsoft Azure Storage [4], and etcd [1]. Both Multi-Paxos and Raft adopt "*full copy*" replication, which replicates a complete copy of data to all nodes.

For modern workloads in highly reliable and available systems, full copy replication is facing performance degradation, mainly due to two reasons: (i) Data size is ever-growing in data-intensive systems. The larger the data size, the larger the storage and network costs when using full copy replication; and (ii) For reliability and durability, many production systems (e.g., etcd and Azure) choose to flush data to disk, before acknowledging a write (or an update) operation. Disk I/O quickly becomes the performance and scalability bottleneck.

More concretely, writing $M$ units of data into the system requires $M$ units of storage space and incurs $M$ units of disk I/O on *each* node. For an $n$-node leader-based SMR (such as Multi-Paxos and Raft), the leader node needs to perform $M * n$ units of network I/O.

Recent SMR systems, such as RS-Paxos [11] and CRaft [17], adopt erasure coding to reduce costs, and show improved performance. However, combining leader-based SMR and erasure coding brings two main practical limitations: (i) During leader failover, the new leader needs to perform recovery (which reconstructs each piece of data from coded segments). Such a recovery workload is proportional to the size of the entire system and takes much longer than recovery in full-copy protocols. No operation can be performed during the recovery, causing a loss of availability; and (ii) In the phase of a slow or saturated leader, the read availability is impacted.

Motivated by the observations, this paper aims to answer: "*How do we build an erasure coding SMR that improves performance and availability, by further alleviating the bottlenecks at the leader*?" In particular, we share our experience in (i) combining erasure coding with a recent *leaderless* SMR system, Rabia [13]; (ii) integrating our design "Racos (RAbia COded System)" with etcd [1]; and (iii) evaluating Racos under realistic workloads using YCSB [6].

## 2 WHY LEADERLESS ERASURE CODING SMR?

Erasure coding within individual datacenters has been widely adopted in production cloud storage and file systems, e.g., Microsoft Azure [4] and Facebook's Tectonic Filesystem [14]. This is mainly because erasure coding allows the systems to explore the tradeoff among storage cost, availability, and reliability. Such an economic argument and operation flexibility are appealing to cloud providers in the era of ever-increasing data volume and velocity.

These cloud systems are designed to handle larger data or files in the range of several megabytes (MBs) or more (e.g., blob or data warehouse storage). These systems typically implement state

*This material is based upon work partially supported by the National Science Foundation under Grant CNS-2238020.

machine replication (SMR) and erasure coding separately. In other words, SMR is a standalone component.

**Prior Erasure Coding SMRs.** It was not until recently that erasure coding SMR was introduced to handle smaller data, from kilobytes (KBs) to MBs. The novelty is on the *integration* of erasure coding with an SMR protocol. RS-Paxos [11] integrates erasure coding with Paxos. CRaft [17] integrates coding with Raft. Both systems outperform the original protocols in local-area networks. Subsequent works improve on various aspects [9, 16, 18].

All these erasure coding SMR systems adopt a leader-based solution; hence, these systems suffer the limitations mentioned in Section 1. While these systems have generally stable and good performance in the common case when the leader is responsive, both performance and availability are impacted when the leader is saturated or even crashed. As availability is the first-class citizen in almost all modern cloud systems, these erasure coding solutions are not as appealing for production systems. This motivates us to explore a *leaderless* design.

All these systems use Reed-Solomon (RS) codes [8], which we also adopt. In particular, the RS code has two parameters $k$ and $m$, both positive integers. In using a $(k, m)$ code, each piece of data is divided into $k$ "data segments" with equal sizes. These $k$ data segments can be used to generate $m$ "parity segments," through *encoding*. In total, there are $k + m$ segments such that any $k$ out of these $k + m$ segments (namely coded segments) are sufficient for recovering the original data, through *decoding*. In our design, each node is assigned to store at least one single data or parity segment; hence, $k + m = n$, which is the number of nodes in the SMR system.

**Towards a Leaderless Erasure Coding SMR.**

Consider erasure coding SMRs that use an $(k, m)$ Reed-Solomon codes and each write operation writes 1 unit of data. In this case, each data or coded segment has a size of $1/k$ units. Prior leader-based systems (e.g., [9, 11, 17, 18]) has storage cost and disk I/O equal to $M/k$ for the follower and $M$ for the leader. This is because the leader stores the full copy of the data (for the purpose of serving reads). For each write operation, the leader needs to obtain the original data from the client and forward the coded segment to a follower, resulting in $M(1 + \frac{n-1}{k})$ network costs.

We choose Rabia [13] as our base SMR protocol; hence, we name our system Racos (RAbia COded System). Each replica in Racos has storage cost and disk I/O equal to $M/k$. The workload of forwarding segments and communicating with clients are distributed evenly; thus, the cost is $\frac{M}{n}(1 + \frac{n-1}{k}) + \frac{M(n-1)}{n}\frac{1}{k} = \frac{M}{n}(1 + \frac{2(n-1)}{k})$.

In addition, the following reasons justify the integration of erasure coding with a leaderless SMR protocol:

- *Performance*: In our targeted scenarios, the nodes have to flush data to disk, resulting in disk I/O being the main bottleneck. The analysis above shows a clear advantage for a leaderless design for writes. Our evaluation also confirms this analysis. Compared to the closest competitor (RS-Paxos [11]), Racos has a 3.36x throughput improvement in all-write workload and 1.6x improvement in the workload with balanced reads and writes.
- *Availability*: The liveness of leader-based systems depends on the responsiveness of the leader. In addition, the expensive failover of leader-based erasure coding SMR brings a concern of unavailability. Finally, since Rabia is based on an asynchronous consensus protocol, reads can be made non-blocking, using a quorum-based protocol, which is generally not easy for leader-based SMR.
- *Development*: Another benefit is the lower development effort in building a complete system. Again, this is due to the complication of failover. It is non-trivial to recover from a leader failure, which requires leader election, fetching coded segments, and decoding to recover the original data, etc. In comparison, Rabia does not require a failover due to its leaderless design.

## 3 RACOS: DESIGN AND EVALUATION

Due to page limitation, we only briefly discuss our key designs, without the full specification. Rabia [13] is a recent leaderless SMR system that builds on top of Ben-Or's binary agreement protocol [3]. Rabia consists of (i) a spreader that exchanges client requests; and (ii) a consensus component that takes 1.5 RTTs to complete on the fast path and 5 RTTs on average. Rabia is leaderless; thus, each replica can serve client requests directly.

Since the consensus component relies on Ben-Or's randomized protocol and the usage of a common coin, Rabia may take more than 1.5 RTTs to agree on a request for each slot (of the SMR log). It was shown in [13] that Rabia outperforms Multi-Paxos in a 3-node cluster when network is well-behaving (i.e., delivering messages in roughly the same order).

*Practical Concerns and Challenges.* As pointed out in [2], there are a few concerns about deploying Rabia under practical workloads: (i) Rabia does not have a natural support for pipelining and only shows a good performance when using batching. However, in our targeted scenarios, batching is not a viable solution for larger data; (ii) Rabia only has a favorable or competitive performance in a 3-node cluster with a well-behaving network; and (iii) Rabia requires more messages and rounds even on fast-path, and may derail from it under high workload.

These limitations are mainly because that Rabia's design requires all-to-all communication and its usage of randomization to break a tie. Our main intuition behind the design of Racos is that using erasure coding, performing all-to-all communication is no longer a bottleneck. In addition, with larger data, conflict is less likely, which means that Racos is mostly on fast-path (without using randomization to break a tie).

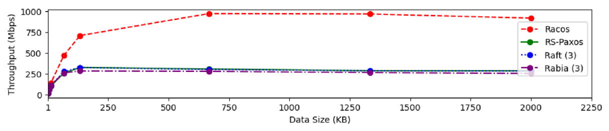
*Racos: Our Design.* We first change the spreader component. Instead of the full data copy, the spreader sends a corresponding coded segment to each node, upon receiving a write request from a client. It was designed in a way that only the segments that are received by a sufficient number of nodes (i.e., $k + f$) will be used as candidates for the consensus component. We then use the same architecture of the consensus component (i.e., Ben-Or and common coin), except that we change the threshold from $f+1$ in Rabia to $k+f$. This ensures that all the committed writes are decodable for future reads, even if $f$ nodes crash. As a result, Racos guarantees liveness if $n \geq 2f + k$, which achieves the same resilience for RS-Paxos.[1]

[1]CRaft [17], HRaft [9], and FlexRaft [18] improve the resilience; however, they assume a perfect failure detector, which is usually not available in real systems.

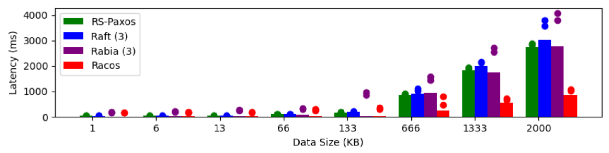
We also present two practical optimizations to further improve performance and scalability over the original Rabia:

- We reduce the fast-path latency of the consensus component from 1.5 RTTs to 0.5 RTTs. In the exchange phase, as long as nodes observe $n - f$ identical requests, then they can immediately agree, *without* entering the Ben-Or phase. This is possible whenever $n \geq 3f + 1$, since enough redundancy ensures that any node in the Ben-Or phase must agree on the same value (for the SMR slot).
- We increase the chance of staying on the fast path by using a quorum-based approach for read operations.

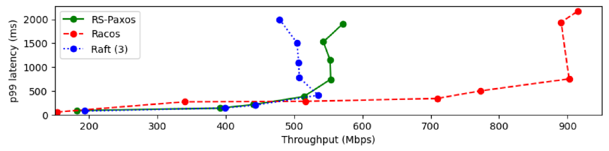
**Evaluation.** We evaluate Racos by comparing it against three other systems Raft [12] (the original SMR used by etcd), Rabia [13] (our base SMR system), and RS-Paxos [11]. RS-Paxos has the best performance in the common case, compared to other leader-based SMRs [9, 17, 18]. We implement Rabia, RS-Paxos, and Racos into etcd, by following the implementation of Raft as closely as possible, e.g., using a similar pattern to exchange messages and accessing disks.



**(a) *All-write workload.* Throughput across different data sizes, ranging from 1.3 KBs to 2 MBs.**



**(b) *All-write workload.* Median latency (bar) and tail latencies (p95/p99).**



**(c) *YCSB-A: 50% reads and 50% writes with data size* = 1.33 *MBs.***

Our cluster consists of 8 nodes running Ubuntu 20.04.5 LTS. Each node is equipped with two 2.4 GHz 64-bit 8-Core CPUs, 64 GB RAM, and one 200 GB SSD. The network bandwidth between each pair of nodes is measured at 9.41 Gbits/s, using iperf. 1 RTT is 0.125 ms. In our experiments, we have 5 or 7 server nodes that run etcd and 1 client node that runs Go-YCSB, which is Go port of YCSB [6], to simulate realistic workloads that follow Zipfian distribution.

We first measure a favorable setup for Racos, all-write workload. In this case, the disk I/O is the bottleneck, even for data as small as 1.3 KBs. Figure (a) presents the throughput (in Mbps) across different data sizes. Racos and RS-Paxos have $n = 5$ and use (3, 2)-RS code, whereas Raft and Rabia have $n = 3$. All systems tolerate 1 crash. Racos has 3.36x throughput improvement over RS-Paxos when the systems saturate, mainly because Racos writes less into disk. Figure (b) shows the median, 95%tile (p95), and 99%tile (p99) latencies (in ms). For smaller data, Racos and Rabia suffer from a higher tail latency due to the usage of Ben-Or and the lack of pipelining. However, as data size increases, the tail latency improves, due to Racos's lower disk I/O cost.

Figure (c) presents the throughput-latency plot using a balanced workload (YCSB-A). We vary the number of client threads from 1 to 50. The throughput advantage of Racos shrinks to 1.6x, mainly due to two reasons: (i) RS-Paxos assumes a leader lease; hence, the leader can immediately return the data without doing any communication; and (ii) Racos needs to obtain coded segments from other replicas for decoding. Racos has a slightly higher p99 latency.

The following table measures Racos's throughput (in Mbps) under different scenarios with all-write workloads. We choose this workload to stress the consensus component, as our quorum-based reads can reduce the contention. For $n = 5$, we compare the case when the message drop rate (on each link) is somewhat aggressive at 0.01%. For $n = 7$, we test both (4,3) and (3,4) erasure codes. The (4,3) code tolerates 1 fault, whereas the (3,4) code tolerates 2 faults.

| data size | n=5 | n=5; 0.01% drop | n=7; (3,4) code | n=7; (4,3) code |
|---|---|---|---|---|
| **1.3 KB** | 17.36 | 8.62 | 18.41 | 19.17 |
| **2 MB** | 916.8 | 918.4 | 923.2 | 1233.6 |

Racos suffers performance loss with smaller data and 0.01% message drop rate. Interestingly, the impact of message drop is negligible when data becomes larger. In our experience, message drop is a non-factor when data size is above 500 KB. (Data not shown for lack of space). Racos also scales well with a larger $n$, mainly due to our practical optimizations. Using (4,3) code, Racos has an improved throughput, as the size of the coded segments is smaller (1/4 vs. 1/3). However, this comes with a reduced fault-tolerance.

## REFERENCES

[1] etcd: A distributed, reliable key-value store for the most critical data of a distributed system. https://etcd.io/, accessed May 2021.
[2] Reading group. Rabia. https://charap.co/reading-group-rabia-simplifying-state-machine-replication-through-randomization/, accessed Jan 2024.
[3] M. Ben-Or. Another advantage of free choice (extended abstract): Completely asynchronous agreement protocols. In PODC 1983.
[4] B. Calder et al.. Windows azure storage: A highly available cloud storage service with strong consistency. In SOSP 2011.
[5] Y. L. Chen et al. Giza: Erasure coding objects across global data centers. In ATC 2017.
[6] B. F. Cooper et al. Benchmarking cloud serving systems with YCSB. In SoCC 2010.
[7] J. C. Corbett et al. Spanner: Google's globally-distributed database. In OSDI 2012.
[8] W. C. Huffman and V. Pless. *Fundamentals of Error-Correcting Codes.* Cambridge University Press, 2003.
[9] Y. Jia et al. HRaft: Adaptive erasure coded data maintenance for consensus in distributed networks. In IPDPS 2022.
[10] L. Lamport. The part-time parliament. *ACM Transactions on Computer Systems (TOCS)*, 16(2):133–169, 1998.
[11] S. Mu et al. When paxos meets erasure code: reduce network and storage cost in state machine replication. In HPDC 2014
[12] D. Ongaro and J. Ousterhout. In search of an understandable consensus algorithm. In ATC 2014.
[13] H. Pan et al. Rabia: Simplifying state-machine replication through randomization. In SOSP 2021.
[14] S. Pan et al. Facebook's tectonic filesystem: Efficiency from exascale. In FAST 2021.
[15] F. B. Schneider. Implementing fault-tolerant services using the state machine approach: A tutorial. *ACM Comput. Surv.*, 22(4):299–319, 1990.
[16] M. Uluyol et al. Near-optimal latency versus cost tradeoffs in geo-distributed storage. In NSDI 2020.
[17] Z. Wang et al. CRaft: An erasure-coding-supported version of raft for reducing storage cost and network cost. In FAST 2020.
[18] M. Zhang et al. Minimizing network and storage costs for consensus with flexible erasure coding. In ICPP 2023.