Serverless primitives for the shared log architecture

Stephen Balogh
Stream Store (s2.dev)
stephen@s2.dev

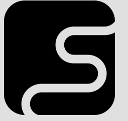
This talk

1. Why data in motion deserves its own first-class, serverless storage primitives

2. What could the next generation of data-intensive applications look like, if we used this primitive as a foundation

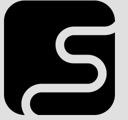
Part I
Serverless streams

Object storage



- Scale
- Performance and reliability
- UI/UX

Object storage



- Material improvement to pre-existing domains
- Durability and scale at the heart of new architectures

Object storage

- Material improvement to pre-existing domains
- Durability and … res

The Rise of Object Storage: Simplifying … Architecture

One of the most significant shifts in modern system design is the growi… storage solutions like Amazon S3. Traditionally, systems managed their… leading to complex challenges around data replication and consistency… emerging approaches to incorporating object storage:

- **Tiered Storage**: The most incremental approach, where less frequently … moved to object storage while maintaining traditional storage for hot dat…
- **Write-Ahead Log Architecture**: A more sophisticated approach where o… (recent mutations) lives outside object storage, while the bulk of data res… like NEON are using this approach to create "bottomless" Postgres imple…
- **Zero Disk Architecture**: The most radical approach, championed by com… Warpstream, where systems abandon local disks entirely in favor of obje…

The implications are profound: simpler operational processes, easier system integration, and greater flexibility in balancing cost, latency, and durability. Perhaps most intriguingly, this shift could fundamentally change how we think about data integration, potentially reducing our reliance on message queues and traditional ETL processes.

S3 as the universal infrastructure backend

Davis Treybig · Follow
Published in Innovation Endeavors · 12 min read · Oct

377 9

TL;DR

1. Traditionally, infrastructure services such as … own storage layer on top of local disk storage… partially a holdover from the pre-cloud era.
2. Increasingly, S3 is being used as the core per… infrastructure services (e.g. Snowflake, Neon… WarpStream), rather than simply as a backup…

KubeCon CloudNativeCon North America 2024

Object Storage Is All You Need

Justin Cormack, Docker

0:01 / 31:41

Object Storage Is All You Need - Justin Cormack, Docker

Why streams?

- Object storage is not directly well suited for frequent writes
- Streams are currently relegated to higher-touch specialist systems
  - Caps on numbers of ordered sequences
  - Low per-stream throughput norms in serverless offerings
    - Use at scale typically demands partitioning
  - High complexity

S2, the Stream Store

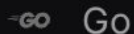
- True serverless API for data in motion, backed by object storage
- **Stream** as an ordered sequence of binary records
- **Basin** as a namespace for streams

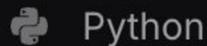
S2, the Stream Store

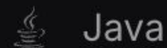
- True serverless API for data in motion, backed by object storage
- **Stream** as an ordered sequence of binary records
- **Basin** as a namespace for streams

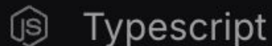
S2, the Stream Store

- True serverless API for data in motion, backed by object storage
- **Stream** as an ordered sequence of binary records
- **Basin** as a namespace for streams

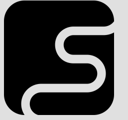








S2: Stream API



Append

Read

CheckTail

S2: Stream API

Append

- Add records to the end of a stream
- All writes are durable within an AZ on acknowledgement
- ~125MiB/s write throughput **per individual stream**
- ~25ms p50 / <50ms p100 writes for Express storage class

Read

- Read records linearly from any point in a stream
- Real-time tailing
- Records from last ~20 seconds at 500MiB/s
- Effectively unlimited throughput for reads of older records

CheckTail

- Retrieve the next sequence number that will be assigned on a given stream
- Single-digit ms operation

S2: Stream API

- Existing “core-streaming” use cases improved
- New patterns for data-intensive architectures

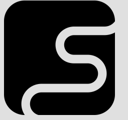
Part II

The shared log abstraction

Shared logs / journals

- WAL on attached-storage typically at foundation of database durability
- If the WAL can be shared among nodes in a fault-tolerant and consistent way, then we can treat storage concerns separate from the state machine
- Architecture at the heart of systems like MemoryDB (AWS), Aurora (AWS), Delos (Meta)

Shared logs / journals

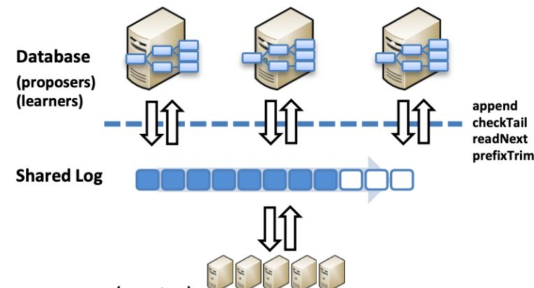


Taming Consensus in the Wild (with the Shared Log Abstraction)

Mahesh Balakrishnan
Confluent, Inc.

Abstract

The shared log is an abstraction for building layered consensus systems that are simple to develop, deploy, evolve, and operate. Shared logs emerged from systems research and have seen significant traction in industry over the past decade. In this paper, we describe some design principles for consensus-based systems, based on our experience building and operating real-world shared log databases in the wild.



Shared logs / journals

“As an analogy, ***think of the [state machine replication] platform as a filesystem and the shared log as a block device****.*

[…] a shared log helps us write an SMR layer without reasoning about the internals of the consensus protocol. The [state machine] layer is then ***free to focus on the complexity of materialization*** *[…]”*

- Mahesh Balakrishnan, “Taming Consensus in the Wild” (2024)

Shared logs / journals

Amazon MemoryDB: A Fast and Durable Memory-First Cloud Database

Yacine Taleb
Amazon Web Services
Canada

Kevin McGehee
Amazon Web Services
USA

Nan Yan
Amazon Web Services
Canada

Shawn Wang
Amazon Web Services
USA

Stefan C. Müller
Amazon Web Services
Canada

Allen Samuels
Amazon Web Services
USA

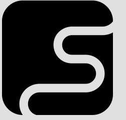
Abstract

Amazon MemoryDB for Redis is a database service designed for 11 9s of durability with in-memory performance. In this paper, we describe the architecture of MemoryDB and how we leverage open-source Redis, a popular data structure store, to build an enterprise-grade cloud database. MemoryDB offloads durability concerns to a separate low-latency, durable transaction log service, allowing us to scale performance, availability, and durability independently from the in-memory execution engine. We describe how, using this architecture, we are able to remain fully compatible with Redis, while providing single-digit millisecond write and microsecond-

and leads to application business logic being customized around the limitations of the underlying storage system.

Open Source Software (OSS) Redis [12], hereafter referred to as Redis, emerged as the most popular in-memory key-value store according to *db-engines.com*[5]. Redis provides microsecond latencies, with p99 under 400us [9]), while allowing applications to manipulate remote data structures, perform complex operations, and push compute to storage. Redis support for complex shared data structures substantially simplifies distributed applications and is chiefly responsible for its popularity. Redis employs asynchronous replication for high availability and read scaling and an on-disk

Shared logs / journals



- But where do you actually find a fault-tolerant, strongly consistent, performant shared log implementation?

Building a KV-store with S2

- Multi-primary, horizontally scalable, distributed KV-store
- All writes regionally durable
- Linearizable reads from all replicas

https://github.com/s2-streamstore/s2-kv-demo

Building a KV-store with S2

- All PUTs and DELETEs add a log entry to a stream (**Append**)
  - Latency bound by S2 stream append end-to-end ack duration
- Replicas all continuously tail the shared log directly, applying events to local materialized state (**Read**)
- All GETs are serviced from materialized state
  - Strong consistency by performing **CheckTail** to ensure local state reflects latest entry in the log

Building a KV-store with S2

- Constraints
    - PUT / DELETE ~ 25ms (median) / 50ms p100
    - Total log append throughput <= 125MiB/s

Building a KV-store with S2

- Concurrency primitives
  - Stipulate an expected next sequence number on append (optimistic)
  - Enforced writer exclusively via fencing tokens (pessimistic)

Building a KV-store with S2

concerns around version upgrades. This is what real system building looks like.

Bonus: Fencing

Above, I mentioned how *fencing* in the journal service API is something that makes the service much more powerful, and a better building block for real-world distributed systems. To understand what I mean, let's consider a journal service (a simple ordered stream service) with the following API:

```
write(payload) -> seq

read() -> (payload, seq) or none
```

You call *write*, and when the *payload* has been durably replicated it returns a totally-ordered sequence number for your write. That's powerful enough, but in most systems would require an additional leader election to ensure that the writes being sent make some logical sense.

We can extend the API to avoid this case:

```
write(payload, last_seq) -> seq

read() -> (payload, seq) or none
```

In this version, writers can ensure they are up-to-date with all reads before doing a write, and make sure they're not racing with another writer. That's sufficient to ensure consistency, but isn't particularly efficient (multiple

Thanks!

s2.dev

stephen@s2.dev

References

- Balakrishnan, Mahesh. “Taming Consensus in the Wild (with the Shared Log Abstraction).” *ACM SIGOPS Operating Systems Review* 58, no. 1 (August 14, 2024): 1–6. https://doi.org/10.1145/3689051.3689053.
- Bhushan, Shikhar. “The Disaggregated Write-Ahead Log.” unofficial blog, November 8, 2023. https://blog.schmizz.net/disaggregated-wal.
- Brooker, Marc. “MemoryDB: Speed, Durability, and Composition.” *Marc’s Blog* (blog). Accessed May 6, 2024. https://brooker.co.za/blog/2024/04/25/memorydb.html.
- *Object Storage Is All You Need - Justin Cormack, Docker*, 2024. https://www.youtube.com/watch?v=ei0wwTy6_G4.
- Riccomini, Chris, and Steven Johnson. “Four Infrastructure Trends Reshaping Modern Systems.” Accessed February 25, 2025. https://www.prefect.io/blog/four-infrastructure-trends-reshaping-modern-systems.
- Taleb, Yacine, Kevin McGehee, Nan Yan, Shawn Wang, Stefan C Müller, and Allen Samuels. “Amazon MemoryDB: A Fast and Durable Memory-First Cloud Database,” 2024.
- Treybig, Davis. “S3 as the Universal Infrastructure Backend.” *Innovation Endeavors* (blog), March 25, 2024. https://medium.com/innovationendeavors/s3-as-the-universal-infrastructure-backend-a104a8cc6991.
- Verbitski, Alexandre, Anurag Gupta, Debanjan Saha, Murali Brahmadesam, Kamal Gupta, Raman Mittal, Sailesh Krishnamurthy, Sandor Maurice, Tengiz Kharatishvili, and Xiaofeng Bao. “Amazon Aurora: Design Considerations for High Throughput Cloud-Native Relational Databases.” In *Proceedings of the 2017 ACM International Conference on Management of Data*, 1041–52. Chicago Illinois USA: ACM, 2017. https://doi.org/10.1145/3035918.3056101.