Introducing RAW Hollow: An In-Memory, Co-Located, Compressed Object Store with Opt-In Strong Consistency

Govind Venkatraman Krishnan
gvk@netflix.com
Netflix Inc
Los Gatos, California, USA

Eduardo Ramirez
eduardor@netflix.com
Netflix Inc
Los Angeles, California, USA

Drew Koszewnik
dkoszewnik@netflix.com
Netflix Inc
Los Gatos, California, USA

Yujia (Cynthia) Xie
yujiax@netflix.com
Netflix Inc
Los Gatos, California, USA

Tej Vepa
tvepa@netflix.com
Netflix Inc
Los Gatos, California, USA

Bernardo Gomez Palacio
bernardog@netflix.com
Netflix Inc
Los Gatos, California, USA

Abstract

In recent years, significant advances have been made to NoSQL databases, enabling them to operate at extreme scale - Cassandra [16] has been benchmarked at over a million writes/second [6]. However our experience at Netflix has shown that applications often deal with small-medium datasets that change less frequently. For these scenarios, while SQL databases remain the popular persistence option [7], we've found that it can be challenging for applications to achieve consistent performance with low latency and high availability. Supplementing the primary store with caching improves read performance, but at the cost of dealing with cache synchronization complexities. Even fully in-memory databases carry the overhead of network calls for every request.

This paper introduces RAW Hollow, a first of its kind object store that is local and in-memory with opt-in strong consistency. With RAW Hollow, the entire dataset is disseminated across the application cluster and resides in the memory of each application process. By leveraging compression, despite being in-memory, RAW Hollow allows scaling datasets up to a 100 million records per entity. Reads are eventually consistent by default, and completely local to the application, ensuring extremely low latencies and high availability. RAW Hollow allows applications to query millions of records in milliseconds. Writes are reflected throughout the cluster with single digit millisecond latencies. Users can select strong consistency at the individual request level allowing them to balance high availability with strong consistency. RAW Hollow simplifies building stateful applications that are highly available and easy to scale.

For this demo, we will load the entire public IMDb dataset into RAW Hollow and allow users to experience first hand the in-memory performance of RAW Hollow by querying, modifying and cloning the data. We will also showcase its resiliency to distributed failures.

SIGMOD-Companion '25, Berlin, Germany

ACM ISBN 979-8-4007-1564-8/2025/06
https://doi.org/10.1145/3722212.3725109

CCS Concepts

• **Information systems** → **Distributed storage**; *Cloud based storage*; • **Computer systems organization** → Dependable and fault-tolerant systems and networks.

Keywords

Distributed Systems, In-Memory Databases, Object Store, Co-located, Data Compression, High Availability, Low Latency

ACM Reference Format:
Govind Venkatraman Krishnan, Eduardo Ramirez, Drew Koszewnik, Yujia (Cynthia) Xie, Tej Vepa, and Bernardo Gomez Palacio. 2025. Introducing RAW Hollow: An In-Memory, Co-Located, Compressed Object Store with Opt-In Strong Consistency. In *Companion of the 2025 International Conference on Management of Data (SIGMOD-Companion '25), June 22–27, 2025, Berlin, Germany*. ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3722212.3725109

1 Introduction

Netflix is the world's largest subscription streaming service with over 300 million users. In order to deliver an exceptional experience to these users, services reliably handle billions of requests daily [17], ensuring low latency and high availability. While Netflix processes petabytes of data each day [17][5], many of its stateful services that are crucial to the user experience manage relatively modest datasets. A prime example is the Video Metadata Service (VMS) [19], which provides metadata for the movies and TV shows available on Netflix. The scale of this metadata is directly correlated with the size of the content catalog, and Netflix's catalog size is orders of magnitude smaller than platforms with user generated content like TikTok or YouTube [18][10][4]. Consequently, services like VMS focus on efficiently handling smaller datasets to be both reliable and scalable.

This need led to the development of Hollow, a total, high density near-cache solution [12]. Hollow employs compression and memory pooling techniques, enabling applications to cache and query their entire dataset in main memory with minimal heap pressure. As a near-cache, Hollow simplifies scaling and ensures high availability by eliminating external dependencies for request processing. Additionally, Hollow offers tools for operational visibility into datasets, including comprehensive change history, version diffing, and zero-copy cloning. Hollow has been battle-hardened over

more than ten years and is the preferred option for slow changing small to medium datasets at Netflix [15].

In 2023, we embarked on an initiative to extend Hollow's capabilities to use cases beyond read-only, resulting in the creation of RAW Hollow (Read After Write Hollow). RAW Hollow integrates the benefits of Hollow with the robust guarantees of a data store, such as atomicity and tunable consistency. The narrow focus of this system is designed to make it simple to build and maintain scalable stateful services.

In the following sections, we start by offering a glimpse into the impact of RAW Hollow within Netflix. We then provide an overview of its architecture and characteristics. Finally we describe the interactive demonstration.

2 RAW Hollow at Netflix

In just over a year since its creation, RAW Hollow has reached over 500 deployments, including over 160 production deployments that include critical tier 0 services. Select examples below.

OCP Live. Open Connect is Netflix's Content Delivery Network. RAW Hollow is used to store network metadata by multiple services within the Open Connect Control Plane (OCP). RAW Hollow allows Control Plane instances to rapidly initialize and store complex in-memory data structures that reflect the network topology, enabling the control plane to satisfy the low latency requirements of live streaming.

OneID. Netflix's internal Identity and Access Management platform OneID uses RAW Hollow to represent its Universal Identity Directory. The high availability coupled with the extremely low latencies for even complex reads allowed OneID to go, with some ingenuity, from a multilayered persistence stack comprising of Cassandra, ElasticSearch [8] and NeptuneDB [3] to just RAW Hollow, and improve their latencies in the process.

Tudum CMS. RAW Hollow powers the Content Management System for Tudum.com, the official companion site to Netflix. Tudum CMS built a WYSIWYG live editor powered by RAW Hollow. RAW Hollow replaced their previous solution which was a combination of a 3rd party CMS provider and a distribution system.

3 System Overview

In the following sections, we present an overview of the Hollow architecture and explain how we have built upon it to develop RAW Hollow.

3.1 Hollow Architecture

Hollow is a Java library and toolset designed to efficiently distribute in-memory datasets from a single producer to multiple consumers, enabling high-performance read-only access. It reduces memory usage by using techniques like encoding, bit packing, and de-duplication, allowing larger datasets to be stored in memory [11].

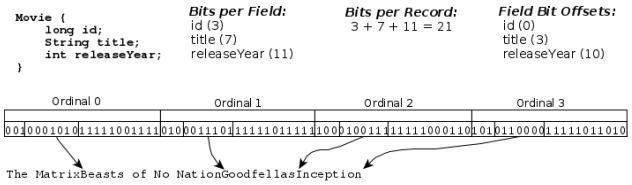

Figure 1: Hollow Example Memory Layout

On the data production side, Hollow serializes the latest and complete state of the dataset into a de-duplicated, highly compact blob, then calculates the changes, known as deltas, since the last state and publishes it to the clients using Gutenberg, Netflix's Pub/Sub service [14]. Under the hood, Gutenberg uses Amazon S3 to store the Hollow data in a versioned manner.

On the data consumption side, every instance has a Hollow Consumer that retrieves the deltas to the local system. The consumer transparently updates the near-cache and makes the new data available to query.

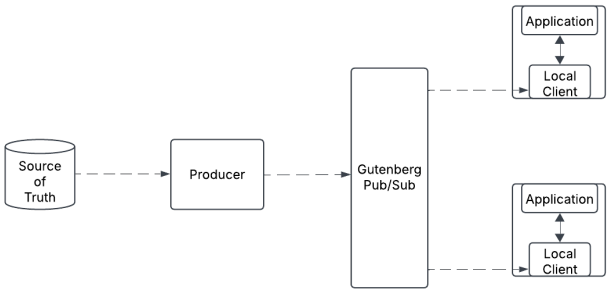

Figure 2: Hollow Architecture

3.2 RAW Hollow Architecture

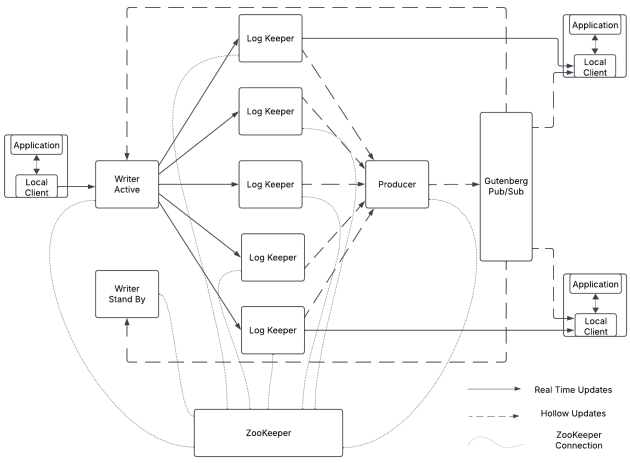

Figure 3: RAW Hollow Architecture

In RAW Hollow, each dataset is identified by a namespace. RAW Hollow follows a single tenant model with each namespace served by its own RAW Hollow deployment. A deployment consists of three distinct instance roles: writers, logkeepers, and producers. In addition to these instance roles, ZooKeeper [13], Gutenberg, and local clients also participate in the system as shown in Figure 3.

Writer. A RAW Hollow deployment will contain multiple writer instances with one of them being active at a time. We use ZooKeeper for leader election. An active writer instance contains a Hollow Consumer that wraps the base dataset, an internal hash table of in-flight changes, as well as a message queue. Each incoming write request is assigned a monotonically increasing offset and added to the message queue. The contents of the message queue are synchronously replicated to all available logkeepers. The writer acknowledges the write with the client only after all available logkeepers commit a message. We have benchmarked the writes on our system to handle a 1000 writes/second with payloads that are 10 KB large when serialized.

Logkeepers. A logkeeper deliberately has very few moving parts. It is implemented as an in-memory circular log statically configured with a size of 1 GB. When a logkeeper receives a message and its offset from the writer, it will append the message to its log and confirm. This operation requires no additional memory allocation, instead, the logkeeper simply updates its internal pointers. Unlike other distributed data stores, since the local clients handle all the read operations, the logkeepers do not need to scale linearly with the application read RPS. The resources for the logkeeper cluster is hence modestly allocated. The writer coordinates the logkeeper quorum via ZooKeeper. If any logkeeper becomes unavailable, the quorum is updated before the next write is acknowledged. This along with the synchronous replication ensures that every individual logkeeper that is part of the quorum is always strongly consistent. Writes fail if the logkeeper quorum falls below the minimum configured threshold.

Producer. Apart from pulling the latest log entries from the logkeeper, the producer behaves very similarly to a regular Hollow producer. The producer updates the base Hollow dataset which propagates via Gutenberg to the writer and all the local clients. When the writer and client receive a new Hollow update, they can reset their in-flight cache. At this point, the writer also triggers the logkeepers to reset their internal log pointers. Zookeeper helps ensure that there is no more than one active producer in a RAW Hollow deployment.

Local Clients. With RAW Hollow, any application instance can become a client of the system and receive an in-memory materialized view with real-time updates for a dataset. Like the writer, a RAW Hollow local client also contains a Hollow Consumer and the hash table of in-flight changes. All the instances keep themselves constantly up-to-date with the logkeepers through a form of long polling. RAW Hollow generates a Java API for the local client which also allows clients to add custom indexes for arbitrary access to any field.

4 RAW Hollow ACID and CAP Properties

4.1 ACID Properties

Atomicity. RAW Hollow has a Bulk Update API that supports grouping operations together atomically, allowing the entire set to all be applied successfully together. If any of the operations fail, the entire set of operations will fail together.

Consistency. The datasets in RAW Hollow always move from one consistent state to another. The Conditional Bulk Update API allows users to specify expectations and ensures that operations will only be executed if the expectations are met. RAW Hollow has a strong schema and all operations are validated against the schema. RAW Hollow supports validation on Primary keys but does not support foreign keys.

Isolation. RAW Hollow provides a single isolation level of Read Committed whereby only committed changes are visible outside the scope of a transaction.

Durability. In RAW Hollow, acknowledged in-flight changes are always retained. Before a write is acknowledged by RAW Hollow, it is synchronously replicated to all available logkeepers. If a writer fails and an election is triggered, the incoming writer has to first connect to the quorum of logkeepers and gather all the in-flight changes. On the other hand if any logkeeper was to restart, or a new one tries to join the quorum, they talk to the existing logkeepers and rapidly collect any in-flight messages before officially joining the quorum.

The logkeepers keep the in-flight messages in main memory, but they are built to be simple and reliable. Logkeepers are also split across all availability zones of a particular AWS region. For all of the logkeepers to go down, the entire region would have to be affected, which is highly unlikely [1]. Additionally, every 30 seconds the producer creates a full snapshot of in-flight changes and uploads them to Gutenberg, which under the hood uploads all the data to S3. Beyond this time period, the full dataset along with the history of changes will be permanently available in S3 which advertises 11 nines of reliability [2]. This durability has been acceptable to tier 0 services in Netflix.

4.2 CAP Properties

By default RAW Hollow is an AP [9] system. Since each local client has an entire copy of the dataset, it is highly available and extremely fault tolerant. Under steady state, the propagation latency to achieve eventual consistency is in the low milliseconds. RAW Hollow also allows users to pick strong consistency at the granularity of individual requests. For those requests, RAW Hollow becomes a CP system. Note that RAW Hollow's implementation of strong consistency still allows users to benefit from the locality of the dataset. When the client receives a request for strong consistency, it holds the request to ensure that it is fully caught up. Subsequent calls within that same scope to the client continue to be local, amortizing the overhead across all of the calls in that request.

5 Demonstration

For the demonstration, we will set up a RAW Hollow cluster and load the entire publicly available IMDb non-commercial dataset[1].Users will access a data explorer UI to experience RAW Hollow's in-memory performance. Figure 4 shows a screenshot of the UI displaying records from the IMDb dataset's Title Basics entity.

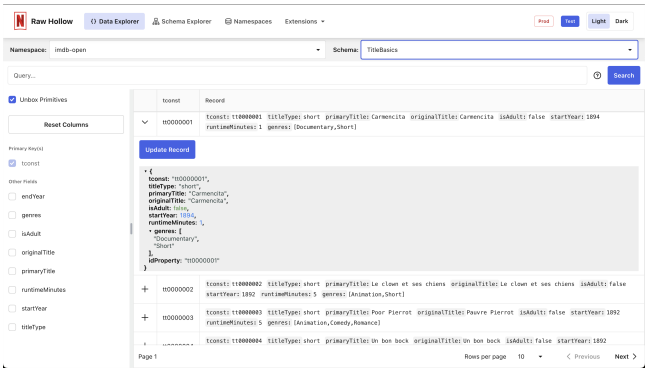

Figure 4: RAW Hollow Data Explorer UI Showing Title Basics Entity from the IMDb Dataset

- **Zero Copy Clone/Restore:** We will showcase RAW Hollow's operational strengths by demonstrating the zero copy clone/restore feature. Users can select an empty RAW Hollow namespace and populate it with the full IMDb dataset. They will observe the memory usage required to store the dataset and can modify the data. At the session's end, the namespace will be restored to its original state.
- **Query Performance:** Users will explore RAW Hollow's query capabilities by scanning thousands of records, retrieving data using unique and hash indexes, or locating records by primary key through the UI. They can update data and see immediate changes with consistent queries. We will demonstrate concurrency by allowing multiple users to modify the same record simultaneously.
- **Resilience to Distributed Failures:** We will demonstrate RAW Hollow's resilience by performing destructive actions on cluster components during use. Figure 5 shows a healthy RAW Hollow cluster. Attendees will observe how the cluster responds to failures and self-heals while continuing to serve requests. Users will see a UI with three counters for a single numerical record: the first represents successful write increments, the second reflects real-time consistent reads, and the third shows eventually consistent reads. We will simulate failures by disabling a logkeeper, showing all counters continue to function. Next, we will disable the active writer, allowing the backup writer to take over. This will result in a brief pause of the write counter, with other counters unaffected. Finally, dismantling the entire cluster will show that while write and consistent read counters fail, the eventually consistent read counter continues, highlighting RAW Hollow's extreme availability.

[1]Information courtesy of IMDb (https://www.imdb.com). Used with permission.

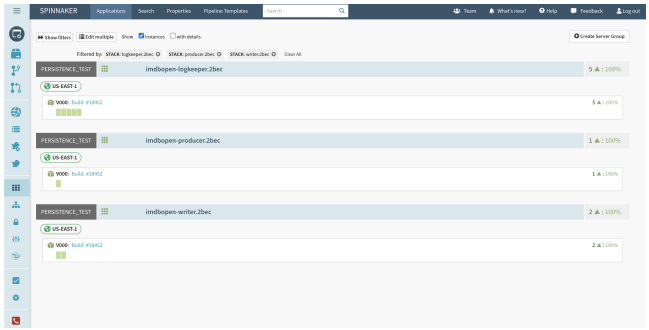

Figure 5: A RAW Hollow Deployment for the IMDb Dataset

References

[1] aws.amazon.com. 2025. *AWS Availability Zones.* Retrieved January 30, 2025 from https://docs.aws.amazon.com/whitepapers/latest/aws-fault-isolation-boundaries/availability-zones.html

[2] aws.amazon.com. 2025. *Data durability and HA with persistent storage.* Retrieved January 30, 2025 from https://docs.aws.amazon.com/whitepapers/latest/real-time-communication-on-aws/data-durability-and-ha-with-persistent-storage.html

[3] aws.amazon.com. 2025. *What Is Amazon Neptune?* Retrieved January 30, 2025 from https://docs.aws.amazon.com/neptune/latest/userguide/intro.html

[4] Matic Broz. 2024. *How many videos are on YouTube in 2025?* Retrieved January 30, 2025 from https://photutorial.com/how-many-videos-on-youtube/

[5] Tianlong Chen and Ioannis Papapanagiotou. 2020. *Bulldozer: Batch Data Moving from Data Warehouse to Online Key-Value Stores.* Retrieved January 30, 2025 from https://netflixtechblog.com/bulldozer-batch-data-moving-from-data-warehouse-to-online-key-value-stores-41bac13863f8

[6] Adrian Cockcroft and Denis Sheahan. 2011. *Benchmarking Cassandra Scalability on AWS — Over a million writes per second.* Retrieved January 30, 2025 from https://netflixtechblog.com/benchmarking-cassandra-scalability-on-aws-over-a-million-writes-per-second-39f45f066c9e

[7] db engines.com. 2025. *DB-Engines Ranking.* Retrieved January 30, 2025 from https://db-engines.com/en/ranking

[8] elastic.co. 2025. *Elastic Documentation.* Retrieved January 30, 2025 from https://www.elastic.co/docs#browse-all-docs

[9] Seth Gilbert and Nancy A. Lynch. 2002. Perspectives on the CAP Theorem. *Massachusetts Institute of Technology* (2002). https://dspace.mit.edu/bitstream/handle/1721.1/79112/Brewer2.pdf

[10] GilPress. 2023. *Tiktok Statistics For 2024: Users, Demographics, Trends.* Retrieved January 30, 2025 from https://whatsthebigdata.com/tiktok-statistics/

[11] hollow.how. [n. d.]. *Hollow In-Memory Data Layout.* Retrieved January 30, 2025 from https://hollow.how/advanced-topics/#in-memory-data-layout

[12] hollow.how. [n. d.]. *Hollow (Netflix OSS).* Retrieved January 30, 2025 from https://hollow.how/

[13] Patrick Hunt, Mahadev Konar, Flavio P. Junqueira, and Benjamin Reed. 2010. ZooKeeper: Wait-free Coordination for Internet-scale Systems. In *2010 USENIX Annual Technical Conference (USENIX ATC 10).* USENIX Association. https://www.usenix.org/conference/usenix-atc-10/zookeeper-wait-free-coordination-internet-scale-systems

[14] Ammar Khaku. 2019. *How Netflix microservices tackle dataset pub-sub.* Retrieved January 30, 2025 from https://netflixtechblog.com/how-netflix-microservices-tackle-dataset-pub-sub-4a068adcc9a

[15] Drew Koszewnik. 2016. *NetflixOSS: Announcing Hollow.* Retrieved January 30, 2025 from https://netflixtechblog.com/netflixoss-announcing-hollow-5f710eefca4b

[16] A. Lakshman and P. Malik. 2010. Cassandra: a decentralized structured storage system. *SIGOPS Oper. Syst. Rev.* 44, 2 (2010).

[17] Joseph Lynch. 2024. *How Netflix Ensures Highly-Reliable Online Stateful Systems.* Retrieved January 30, 2025 from https://www.infoq.com/articles/netflix-highly-reliable-stateful-systems/

[18] netflix.com. 2023. *What We Watched: A Netflix Engagement Report.* Retrieved January 30, 2025 from https://about.netflix.com/en/news/what-we-watched-a-netflix-engagement-report

[19] Shobana Radhakrishnan. 2013. *Object Cache for Scaling Video Metadata Management.* Retrieved January 30, 2025 from https://netflixtechblog.com/object-cache-for-scaling-video-metadata-management-c3c17830983e