Mangalayatan Journal of
Scientific and Industrial Research

Man. J. Sci. Ind. Res.
Volume 2 (1) (2025), 25203
ISSN: 3049-2815

Article

Timestamp-Based Validation Approach for Detecting Stale Data in Asynchronous Microservice Architecture

Sandeep Sharma[1] and Vijay Pal Singh[2]

[1] *Department of Computer Engineering & Applications, Mangalayatan University, Aligarh*
[2] *Department of Computer Engineering & Applications, Mangalayatan University, Aligarh*
***Corresponding Author: 20230102_sandeep@mangalayatan.edu.in*

Abstract

Data freshness is a crucial problem for microservice architectures as services are decoupled and produce the effects of updates in an asynchronous manner. In these contexts, addressing the issue of stale data across service boundaries is still a pressing issue. This paper introduces the Timestamp-Based Validation Approach (TBVA), a lightweight method that can be employed to identify stale information without the overhead of strong consistency models or a central coordinating process. With TBVA, services can evaluate data freshness at query time by referring to short-lived temporal metadata updated with recent update activity. This metadata is kept outside in a cache store like Redis, administrated by expiration policies to balance performance and reliability. Services detect and mark potentially stale data by comparing the cache timestamp with the record's modification timestamp in the database before returning responses. The technique increases the integrity of cross-service data and allows responsiveness and architectural freedom. TBVA is particularly suitable for event-driven, asynchronous microservice systems, in which real-time correct processing is critical while achieving global transactional consistency is also not feasible.

Keywords: *Microservices, Data Freshness, Stale Data Detection, Distributed Systems Consistency, Asynchronous Systems, Timestamp-Based Validation*

1. Introduction

With the advance of microservice architectures in modern systems engineering, it becomes more common to design for scalability, maintainability, and fault tolerance. Each service works in a decoupled mode, maintaining its own data, talking over event streams or messaging queues with others. This division enables independent functioning, increasing the system's flexibility and facilitating accelerated growth. But microservice systems come with architectural benefits, and they also bring challenges in keeping data fresh and consistent among independently running parts of the system.

The main difficulty in such architectural designs emerges from maintaining data consistency, which requires users to access updated information from other services. Under eventual consistency models and asynchronous communication, services can access old data versions because the data still needs to be updated and distributed throughout the system. Practical issues in e-commerce and real-time analytics and financial systems face major challenges due to their

Received: 7 May 2025; Accepted: 15 May. 2025

dependence on recent data for correct and prompt operations.

Traditional methods for stale data management through quorum-based reads or distributed locks and coordination protocols usually lead to service autonomy restrictions, and loose coupling violations as well as performance slowdowns. The implemented solutions become unnecessary for systems that maintain strong consistency only through staleness detection. The development of simple decentralized data freshness monitoring methods has become necessary because services require real-time freshness checks without relying on blocking operations or centralized oversight.

TBVA represents the Timestamp-Based Validation Approach, which serves as an effective technique for asynchronous microservice environments to detect possible data staleness. TBVA functions by identifying update intention via temporal metadata—specifically, a transient timestamp maintained outside the system in a fast-access cache like Redis. When a service starts an update, the service records a timestamp in Redis associated with the GUID of data. This metadata is short-lived with a time-to-live (TTL), and lives only long enough to justify read operations happening in the near future.

A read action is carried out by the querying service to read a record from its local store and metadata from Redis. Then, the timestamp T_m of update intent is compared with the last-modified timestamp T_d of the data record. If the timestamp from the metadata is newer than the last time the data was written, It demonstrates that the record is potentially stale; otherwise, it's fresh. If no metadata is discovered, we default to assuming freshness to keep things responsive. TBVA is ideal for event driven architectures where services emit update events and data asynchronously persisted. It allows us to validate freshness in real-time without blocking read or couple services too close together. With a simple and efficient mechanism of comparing timestamps, TBVA is well-suited to be a scalable and easy-to-integrate solution to enhance data correctness in distributed systems. This paper provides an in-depth description of the TBVA design, operational logic, implementation trade-offs, and presents a use case along with the architectural diagrams that illustrate its capabilities toward use in actual systems.

2. Related work

Research efforts have continuously targeted maintaining fresh data in distributed systems and microservice architectures since the early years. The shift from monolithic systems to microservices creates major challenges in ensuring that all services work using correct and contemporary information. Researchers have developed several methods to manage consistency constraints, identify dormant data and reduce eventual inconsistency issues between services.

When developers aim to enhance freshness guarantees they turn to the use of quorum-based consistency models which can be found in systems like Amazon Dynamo and Apache Cassandra. These systems allow users to set **W** and **R** replica crossover requirements to adjust their consistency levels. The implementation of quorum techniques works effectively, yet it requires substantial coordination effort between servers thus it fits better in data storage systems, they are unsuitable for microservices that prioritize non-blocking reads and decentralized state management. Additionally, such models can degrade performance in latency-sensitive applications due to their coordination overhead [1].

Version vector and vector clocks allow distributed systems to track causal updates via logical timestamps. Applications using Riak as their key-value store frequently encounter multiple conflicting versions called siblings through their eventually consistent operation. Through dependency tracking COPS continues the model from COPS to establish causal consistency for operations [2]. Both systems become difficult to scale and coordinate when the number of dependent services rises. The expansion of Riak vector clocks becomes unwieldy when many concurrent write operations occur at the same time but COPS mandates service knowledge and operational dependency which disrupts the independent nature of microservices. These systems require uniform versioning and coordination protocols even though this assumption does not match the heterogeneity of cloud-native environments.

Few new models rely on timestamps to validate the data freshness without blocking system operations. For instance, Parekh et al. GA introduced time-based distributed caching with short-living metadata for web systems [3]. But this is at the cache layer, and thus the validation happens before the query goes to the core business logic. This presents the risk of false freshness, as it could be the case that the cached metadata is recent enough to look valid, but the underlying data has grown old because of missed cache updates or considerable re-caching delay.

Cloud-native HTAP systems from the present era place timestamp-based validation logic directly inside their database layer. The integrative nature doubles down on accuracy in freshness validation, although it demands tight application-system integration at the application storage interface. These systems demonstrate restricted portability because they operate poorly in environments filled with decentralized microservices [4].

Serverless and log-structured systems implement timestamped logs to verify and retrieve state information during state recovery operations. The Halfmoon platform uses log-optimal, fault-tolerant methods that depend on timestamps to achieve consistency throughout its asynchronous functions [5]. The asymmetric logging method enhances both execution visibility and decreases coordination requirements [6]. These systems excel at implementing consistency during event replay and fault recovery operations, but provide no solutions for detecting stale data during runtime microservice data reads.

A study implemented timestamp-indexed validation strategies in time-series databases for energy infrastructure by using Apache Druid to deliver fresh-time-aware query capabilities for streams of asynchronous data [7]. This methodology produces successful results in controlled environments but requires specific structured ingesting operations, which lead to generalization problems when working with decentralized microservices. The integration process demands collaboration between data schemas and query planners which reduces the individual autonomy a service system operates with.

None of these approaches offer a low-cost way of verifying data freshness at the time of the query within the application service itself, without the use of synchronized clocks, vector clocks, or cross-service dependency tracking. The Timestamp-Based Validation Approach (TBVA) fills this gap in that it makes use of ephemeral and TTL-bound metadata that a service stores externally (e.g., in Redis) to let other services determine whether the data they just retrieved is stale. As a result, it allows for real-time, decentralized validation without the need

to coordinate, and, as such, is very suitable for asynchronous, event-driven architectures. In contrast to both cache-level and database embedded freshness mechanisms, TBVA operates at the application layer, allowing service autonomy and actionable freshness guarantees.

3. Methodology

Time-Based Validation Approach (TBVA) delivers a flexible and distributed method to detect outdated data throughout asynchronous service frameworks. TBVA operates differently from traditional approaches by giving services the ability to check the freshness of their records with temporal metadata when they query information. The section describes operational workflows alongside component duties, which includes metadata management as well as architectural prerequisites, and real-life usage descriptions, and formal validation logic elements.

3.1 Conceptual Model

All the services in distributed microservice frameworks operate independently with their individual databases, using asynchronous protocols for communication. The choices made in design lead to better scalability but present challenges for maintaining fresh data between different services. The system controls update detection separately from data storage through its TBVA design. The metadata layer in Redis operates as a transient system that tracks update timestamps for data registration. Services can determine the staleness of data through metadata saved without requiring central management.

3.2 Operational Workflow

TBVA functions across two separate operational sequences that include both the write path as well as the read path.

3.2.1 Write Path

1. A command service accepts requests for record adds or updates.
2. A metadata entry comprising the GUID and timestamp $T_{m,}$ is immediately stored in Redis with a Time To Live (TTL).
3. A message queue receives the entire data payload at the same time as metadata entry storage occurs.
4. A subscribed worker service performs asynchronous updates to the service database using timestamp T_d.

3.2.2 Read Path:

1. When users make read requests the query service obtains the record from its database along with updated timestamp (T_d).
2. The service retrieves metadata information from Redis cache based on record GUID fetch from database query.

3. The service conducts a timestamp comparison between $T_{m,}$ which represents Redis metadata timestamp, and timestamp T_d which represents the modified time of database records.
4. If ($T_d >= T_m$ or CacheMetaData not exist) the record is flagged as **fresh**; otherwise, it is considered **stale**.

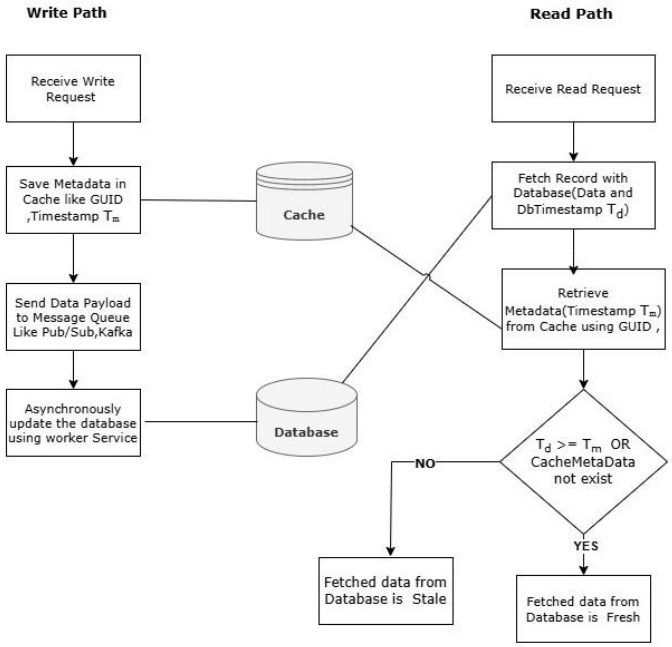

Figure 1: Timestamp-Based Validation Approach (TBVA) Operational Model Workflow.

3.3 System Components

1. **Command Service:** Begins the update operation and records the intent to update.
2. **Query Service** performs the data query and validation of its freshness.
3. **Cache**: Caches short-lived timestamp metadata with TTL, like Redis.
4. **Message Broker:** Update events gets transported asynchronously (E.g.., GCP Pub/sub, Kafka).
5. **Worker Service :**Listens to the broker and update the database.
6. **Service Database:** Persists domain data for every service

3.4 Metadata Management

Redis allows metadata to survive only briefly after setting its TTL parameter length. The system deems metadata inaccessible following its expiration time. During metadata unavailability, the system operates with current data records by accepting specified risks of temporary out-datedness. TTL parameters need adjustment based on the update speed and freshness limits that apply to each service.

3.5. Assumptions

The approach operates under the following assumptions:

1. All participating services maintain **loosely synchronized clocks** (e.g., via NTP).
2. The Redis cache and message broker are **highly available** and resilient.

3.6. Illustrative Use Case

A contemporary e-commerce platform featuring a microservice-based design includes two self-contained services known as the Inventory Service and Order Service. The Inventory Service retains product inventory data, which it updates automatically through warehouse events and order transactions that occur in real time. Orders processed by the Order Service need to check stock levels before accepting customer transactions.

During stock update registrations, such as deliveries or adjustments, the Inventory Service sends events to the message queue and creates simultaneous Redis operations with validity period timestamps. The Worker Service handles the stock data persistence work asynchronously after consuming the original message from the Inventory Database.

The Inventory Service activates the stock update event, which triggers an order for a new customer but the updated stock quantity is not yet written to the database. The Order Service maintains incorrect stock levels because it obtains database information while the database lacks awareness of a pending inventory update, leading to either rejected orders from valid stock levels or events where invalid stock levels indicate surplus inventory.

At stock validation time the Query Component of the Order Service consults Redis through TBVA. During validation, the Order Service checks Redis timestamp (T_m) against Inventory record timestamp (T_d) to determine database data freshness. The service indicates data potential staleness when it detects that the Redis timestamp exceeds the record timestamp in the database. The Order Service can confirm orders conditionally or display a warning after finding this result because it could avoid placing incorrect orders.

TBVA enables distributed services to check and determine data freshness independently of strict consistency requirements or coordinated operations. The approach maintains accurate service interaction through its data detection system, along with maintaining standard e-commerce system scalability and responsiveness values.

3.7. Performance and Scalability

- The O (1) execution time of Redis lookups maintains low latency delays.
- No distributed locks appear in the system design which also eliminates blocking of reads and global coordination techniques.
- Transient metadata entries consume minimal memory space because they are small in size.
- TBVA provides efficient high-speed real-time processing capabilities keeping data freshness validation both accurate and powerful for massive throughput systems.
- To applies TBVA is very easily with the existing microservices since it applies timestamp logic directly on top of the business data models without requiring service modifications.

4.System Architecture

The Timestamp-Based Validation Approach (TBVA) establishes a framework that enables large-scale, decentralized data freshness validation among asynchronous microservice programs. The system depends on CQRS architecture principles, as well as asynchronous communication and temporal metadata to find outdated data instead of demanding strict consistency.

4.1 Architectural Overview

The system contains multiple independent services that connect through the Command Service, Query Service, Worker Service, Cache (Redis), Pub/Sub Queue (RabbitMQ or Kafka) and, Database. The independently scalable services combine through asynchronous methods which enhance fault tolerance, as well as responsiveness and modular design.

1. The Command Service accepts data update requests and simultaneously:
    - Writes update intent (GUID + timestamp) into the Redis Cache.
    - Publishes the full payload asynchronously to the Pub/Sub Queue.
2. The Worker Service consumes messages from the queue and persists updates into the Database.
3. The Query Service handles read operations. It retrieves the latest record from the Database and validates its freshness by comparing the database's last-modified timestamp with the timestamp retrieved from the Redis Cache metadata.

This pattern enables real-time data validation without introducing blocking reads or distributed locks.

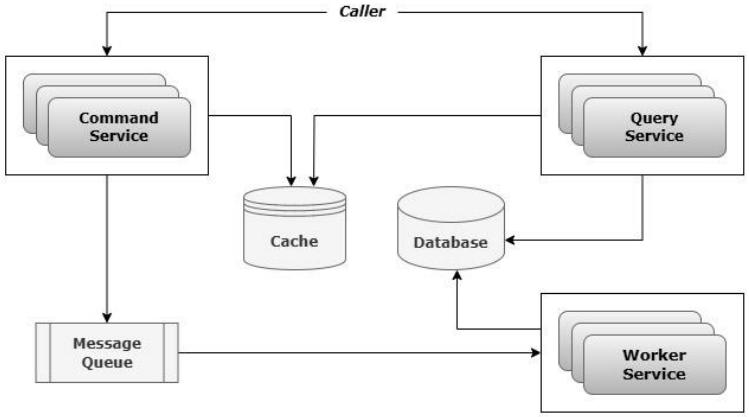

Figure 2: System Architecture of Timestamp-Based Validation Approach (TBVA)

4.2 Operation Flow and Interaction Sequence

The operational workflow is depicted in the sequence diagram below, highlighting the interactions during both the add/update and Fetch operations.

4.2.1 Add/Update Operation

1. The Caller invokes **SaveRecordAsync(GUID, data**) on the Command Service.

2. Two actions happen in parallel:
   - **PushToCacheAsync()** stores metadata (GUID, timestamp T_m) in Redis.
   - **PushToQueueAsync()** sends the data payload to the Pub/Sub Queue.
3. The Worker Service processes the queue message, transforms, and persists the data using SaveRecord() into the Database.

4.2.2 Read Operation

1. The Caller initiates a read via GetRecord(GUID) on the Query Service.
2. The service:
   - Retrieves the record and timestamp from the Database.
   - Fetches the matching metadata from Redis.
3. A comparison is performed:
   - If cacheTimestamp(T_m) is newer than dbTimestamp (T_d), the record is flagged as stale.
   - If Cache Metadata does not Exist, than the record is flagged as fresh.
   - If dbTimestamp (T_d), is newer OR Equal cacheTimestamp(T_m), than the record is flagged as fresh.
   - Otherwise, record is flagged as stale.

The time-stamp verification mechanism grants users both fresh and stale data records with flag to identify the data state.

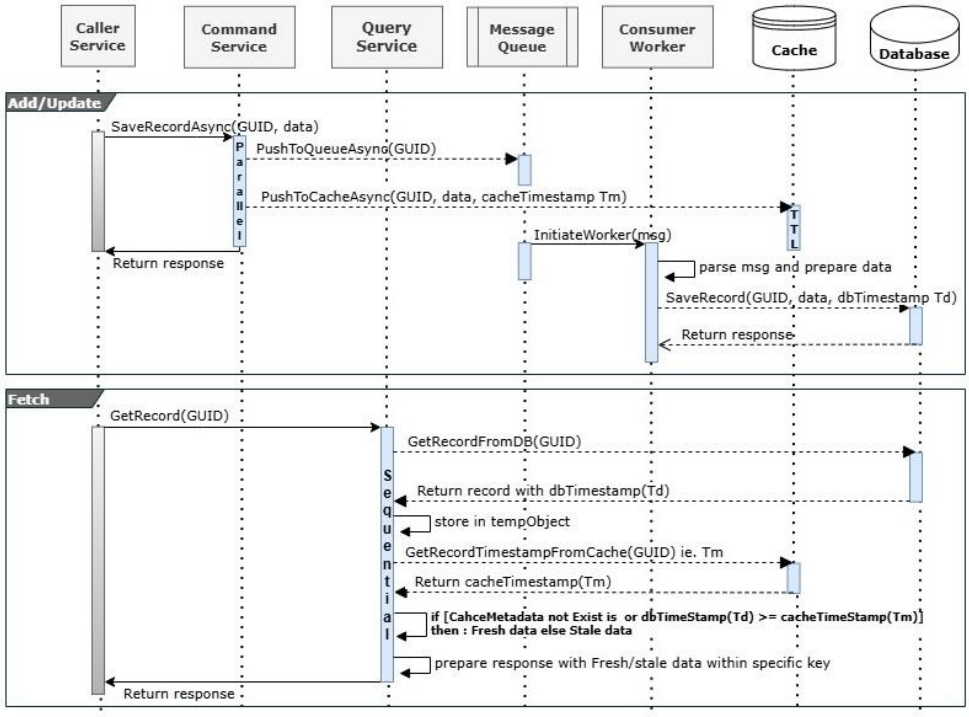

Fig. 3. Sequence Diagram for TBVA-Based Update and Read Operations.

5. Discussion and Limitations

TBVA is a timestamp-based validation method that solves the well-known problem of ensuring that stale data is not used in an asynchronous microservice architecture, whilst not

impacting service autonomy and system performance. By adding a transient metadata tier using in-memory database (Redis), TBVA separates the consistency validation from the propagation of updates, and services can make local data freshness inferences without distributed consensus or strong synchronization mechanisms.

TBVA shows good scalability due to low integration overhead. Each deployment can be deployed independently and does not have a hard dependency on other components or systems in the environment. Since read time complexity of in-memory cache like Redis is O(1), the system is implemented with low latency in situations with a high number of concurrent users. However, there exist some problems or limitations to TBVA. The approach depends on loosely synchronized clocks across services; one with significant drift may cause false positives in freshness evaluation. This can be averted by maintaining time synchronization on all machines using NTP (Network Time Protocol) or similar mechanisms.

Redis metadata storage is the only weak link. If Redis becomes temporarily unavailable, or loses data, services need to believe all database records are fresh which can make it look like there's no staleness.TTL (Time-To-Live) setting is also very important. When TTLs are set too low, metadata may expire before it can be verified, resulting in the missing of staleness detection. On the other hand, using an excessively high TTL could lead to memory overhead and maintaining outdated metadata. Performing effective TTL tuning should take into account domain-specific update rates as well as the desired balance between freshness precision and resource expenditure.

While TBVA works well on the staleness issue, it cannot resolve data conflicts nor maintain causality. In very dynamic systems with fast consecutive updates, other techniques (e.g., version vectors) or conflict resolution mechanisms should be considered to boost TBVA.

In general, TBVA is a lightweight and cost-effective solution for identifying stale data in event-driven, distributed systems. In the future, we should explore adaptive TTL management, more resilient synchronization strategies, as well as large-scale experiments across globally distributed service infrastructures.

6. Conclusion and Future Work

The Timestamp-Based Validation Approach (TBVA) is proposed as a simple option to identify stale data in fully distributed microservice systems. From a TBVA perspective, clients can check the freshness of data independently of the operations on storage, since TBVA decouples the process of checking freshness from that of persistence. Read operations perform constant-time validation using data in Redis cache metadata, together with asynchronous event processing that maintains system scalability along with high performance.

TBVA provides a practical approach in scenarios when full transactional guarantees are impossible, but data freshness is still important. Its merits are low latency, lightweight architectural intrusion, and the retention of microservices' autonomy. Also, using short-lived metadata with expiration policies, the system is both memory efficient and resistant to partial failure. However, TBVA has its limits.It has number of limitations such as dependency on roughly synchronized clocks, vulnerability to metadata TTL tuning and untamed conflict resolution. Future work will investigate the incorporation of logical clock mechanisms in order

to remove physical clock dependencies, exploiting adaptive TTL strategies due to system dynamics, as well as techniques to improve conflict detection and resolution in the presence of highly concurrent operations. In summary, TBVA fills the space between eventual consistency and understanding data freshness in real-time, and offers a scalable and deployable pattern for contemporary event-driven microservice systems.

Reference

1. DeCandia, G., Hastorun, D., Jampani, M., Kakulapati, G., et al. (2007). Dynamo: Amazon's highly available key-value store. Proceedings of the 21st ACM Symposium on Operating Systems Principles (SOSP), 205–220. https://doi.org/10.1145/1294261.1294281
2. Lloyd, W., Freedman, M. J., Kaminsky, M., & Andersen, D. G. (2011). Don't settle for eventual: Scalable causal consistency for wide-area storage with COPS. Proceedings of the 23rd ACM Symposium on Operating Systems Principles (SOSP), 23, 401–416. http://mpaxos.com/teaching/ds/20fa/readings/cops.pdf
3. Parekh, J., Moroney, A., & Golani, L. (2021). A timestamp-based novel caching mechanism for distributed web systems. International Journal of Advanced Computer Science. https://www.researchgate.net/publication/344300301
4. Li, F. (2023). Modernization of databases in the cloud era: Building databases that run like Legos. Proceedings of the VLDB Endowment, 16(13), 4140–4152.
5. Qi, S., Liu, X., & Jin, X. (2023). Halfmoon: Log-optimal fault-tolerant stateful serverless computing. Proceedings of the ACM Symposium on Operating Systems Principles (SOSP), 2023.
6. Qi, S., Feng, H., Liu, X., & Jin, X. (2025). Efficient fault tolerance for stateful serverless computing with asymmetric logging. ACM Transactions on Computer Systems, 2025.
7. Hadjichristofi, C., Diochnos, S., Andresakis, K., & others. (2024). Using time-series databases for energy data infrastructures. Energies, 17(21), 5478. https://doi.org/10.3390/en17215478