ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/25may2034

Kafka Native Delay Subsystem for Distributed Message Processing Preventing Data Loss and Solving Data Residency

Nitin Gupta[1]

[1]CA, USA

Publication Date: 2025/06/05

Abstract: In modern distributed systems, particularly those leveraging asynchronous message processing, the ability to introduce controlled delays for messages is crucial for various functionalities, including retry mechanisms, scheduled deliveries, and rate throttling. This white paper presents the design and operational principles of a novel delay subsystem built entirely on Apache Kafka. By strategically utilizing a set of Kafka topics that represent discrete delay durations, this design eliminates the need for external schedulers, databases, or additional services, thereby minimizing architectural complexity and coupling. It also solves the problem associated with data residency due to various legal concerns in the banking and fintech industry. The paper details the "denomination" approach to accumulating arbitrary delays, analogous to dispensing currency change. It elucidates the inherent advantages of this Kafka-native approach, such as natural sorting, Kafka's robust write performance, and preventing data loss by retaining data temporarily. Furthermore, it provides a comprehensive walkthrough of the message flow, worker behavior, and the critical role of message headers in maintaining logical record integrity. This subsystem offers a highly scalable, resilient, and cost-effective solution for managing delayed messages within a Kafka-centric architecture.

Keywords: *Kafka Delay System; Innovative System;Data Loss Prevention; Data Residency; Banking; Fintech; Data Complaint; Distributed Architecture.*

How to Site: Nitin Gupta, (2025), Kafka Native Delay Subsystem for Distributed Message Processing Preventing Data Loss and Solving Data Residency. *International Journal of Innovative Science and Research Technology*, 10(5), 3546-3550. https://doi.org/10.38124/ijisrt/25may2034

I. INTRODUCTION

In modern distributed architectures, the base of inter-service communication is increasingly asynchronous messaging, predominantly facilitated by high-throughput, low-latency message brokers like Apache Kafka [1]. While Kafka excels at durable message ingestion and scalable distribution, its core design intentionally omits native support for fine-grained message scheduling or arbitrary temporal delays.

However, a critical gap exists for numerous applications for robust and reliable reprocessing of messages that encounter transient failures. For example, a financial transaction system where a downstream service might temporarily be unavailable, or an e-commerce platform where an inventory update fails due to network glitch. In such scenarios, discarding a message in unacceptable. These applications face a significant challenge as they cannot introduce external dependencies, such as relational databases or dedicated queueing systems, solely for the purpose of managing delayed retries. These constraints come from several factors:

A. Data Residency and Compliance:

Storing sensitive message payloads in a separate, potentially less controlled, subsystem can violate stringent legal and data residency regulations (e.g., GDPR, CCPA) [2]. Maintaining an unbroken chain of custody within the Kafka ecosystem is often a paramount requirement.

B. Operational Overhead and Integration Complexity:

Introducing and maintaining another persistent storage layer adds considerable operational burden – requiring separate monitoring, backup strategies, scaling considerations, and integration logic. The goal is to keep the architecture streamlined and minimize points of failure.

C. Performance and Latency:

An external database lookup for every delayed message can introduce unacceptable latency and contention, negating the very performance benefits offered by Kafka.

To precisely address these complex requirements – particularly in environments with strict legal and data residency constraints and where data loss carries severe implications – that a Kafka-native cyclic delay system emerges as an ideal architectural solution. This system effectively transforms

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/25may2034

Kafka itself into a mechanism for managing message retries with precisely controlled delays. By leveraging Kafka's inherent durability and distributed nature, messages that fail processing can be strategically re-enqueued onto internal "delay" topics. The cyclic nature ensures that these messages are re-examined at predetermined intervals and, upon expiration of their delay, are automatically routed back to their original processing topics (or designated "reactivate" topics). This approach allows for:

- *Idempotent Retries:* Messages are re-attempted without introducing duplicates, handling transient issues gracefully.
- *Low Data Loss Risk:* Since the message will be inside the kafka ecosystem,it mitigate the risks associated with data movement to external storage [3].
- *Compliance Adherence:* System operates entirely within kafka environment, so it inherits with data residency and security policies already established with kafka.
- *Simplified Operations:* It avoids maintenance overhead, complexity and integration challenges of managing an entirely separate retry system.

II. ARCHITECTURE AND DESIGN

This system is designed to take advantage of Kafka's strengths without coupling it with additional dependencies and paradigms so it can be extended and reused to various use cases and industries which resemble these criteria. The delay can be applied for multiple reasons, including to:

- *Recoverable:*Retry messages that failed due to a recoverable problem.
- *Scheduling:* Delay messages scheduled for a later time.
- *Throttling:* Throttle the transmission rate.
- *Priority:* Increase the priority of some messages by lowering the priority of others.

The design relies upon a set of topics representing varied durations [4]. We can then enqueue a record into any of the topics any number of times to accumulate a wall-clock delay. For example, if we define topics with these delay durations:

- *Delay-1h*=1 hour delay
- *Delay-5m*=5 minute delay
- *Delay-1m*=1 minute delay

Then a record requiring a delay of 1h19m will be Enqueued as follows:

- *1 time on delay-1h* =1h0m
- *3 times on delay-5m* = 0h5m
- *4 times on delay-1m* = 0h4m

- *Analogy*

This is analogous to the coin dispensing vending machine for dispensing change [5]. For example, to dispense, to dispense 68 cents with the minimum amount of coin, it issues:

- *2 quarters (25 cents each)* = 50 cents
- *1 dime (10 cents each)* = 10 cents
- *1 nickel (5 cents each)* = 5 cents
- *3 pennies (1 cents each)* = 3 cents

The Kafka topics are analogous to the coin Denominations, number of times a record is enqueued into a particular topic is analogous to the number of each coin issued.

III. METHODLOGY

A. Why it works

This technique and design work because of several factors:

- *Sorting occurs naturally:* Every record enqueued into a topic must aawake after the messages preceding it are consumed.
- *Denominations complement each other:* The denominations allow us to reach an accumulated delay with a reasonable amount of queuing.
- *Denominations are adjustable:* We can optimize the denominations as necessary to minimize the number of enqueues or to increase the granularity of delays.
- *Kafka excels at writing:* Kafka's strength is in writing quickly and durably, which lowers the cost of data movement.
- *Follow a logical record:* Multiple physical records represent a logical record in various states of processing via headers.

B. How it works

This example shows how we would handle a record that requires delay, it can fall into the mentioned scenarios, for example, to prevent data loss, to be retried due to system availability, to be throttled or any other.

- The desired wake-up time is added as the header reactivate At.
- The desired revival topic is added as the header reactivate To Topic.
- The delay calculator chooses the maximum delay denomination offered that does not exceed the revival time and writes the future timestamp as the header awakeAt.
- The record is enqueued to the chosen delay topic.
- The delay topic worker dequeues the record and reads the awakeAt 1. If the awakeAt has not yet occurred, the TopiPartitionForePerson pauses the TopicPartition for the time necessary and worker nacks (negative acknowledge) the record so Kafka will relay the offset. 2. After the TopicPartition resumes, we repeat this step.
- Now that the awakeAt has occurred 1. If the reactivateAt has not yet occurred, the delay calculator is invoked again, and we repeat step 3 to further delay the record. 2. If the reactivateAt has occurred, the record is enqueued back to the reactivateToTopic topic.

The minimum delay denomination will always be used when the reactivateAt has not reached to prevent releasing the record back to the reactivate topic too early. This results in additional delay; however, this can be minimized with the inclusion of fine-grained delay denominations.

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/25may2034

If there is a need to reactivate records to receive a higher priority than other records, it can be done by introducing a priority topic and have worker listen to that topic. This idea is analogous to a priority line at a theme park attraction to skip the line [6].

IV. SLEEP VARIATIONS

A record is at the head of the queue when the consumer group offset points to it, i.e., all the earlier records have been acknowledged. When the delay worker calculates topic partition delay duration, it waits for n time units. Three cases need to be handled by delay worker:

- *Delay duration:* When the record is at the head of the queue and nothing is enqueued ahead of it.
- *Zero duration:* When the record is enqueued immediately after another one, and they share the same reactivateAt time.
- *N duration:* When the record is enqueued immediately after another one, but the preceding one has not achieved its awakeAt time.

The system calculates the next delay only when it's needed, which helps compensate for any time lost because of processing delays or system downtime. The following section illustrates in more detail, assuming the topic duration is 2-time units. Record 1 is at the front of the queue and must complete its full waiting time before it's re-processed.

Table 1 Topic delay Duration.

Duration	***Action***	***Queue***
0	- 1 is enqueued - Worker nacks 1 and sleeps for 2-unit time	1 ←head
2	- Worker awakes - Worker forward 1 to its next destination and acks it - Worker idle	1 ←head

Record 1 is at the head of the queue and record 2 is enqueued immediately behind it; both records share the same reactivateAt time. Record 1 must wait the full duration before being reprocessed. While record 2 implicitly waits the same duration and is then explicitly delayed zero-time units.

Table 2 Zero duration.

Duration	***Action***	***Queue***
0	- 1 is enqueued - 2 is enqueued immediately after - Worker nacks 1 and sleeps for 2-unit time	1 ←head 2 ←tail
2	- Worker awakes - Worker forward 1 to its next destination and acks it	2 ←head
2+delta	- Worker forward 2 to its next destination and acks it - Worker idle	Empty

Record 1 is at the head of the queue and record 2 is enqueued sometime after it. Record 1 must wait the full duration before being reprocessed. While record 2 implicitly waits for the part of the duration and is then explicitly delayed for the remaining duration.

Table 3 N duration.

Time	***Event***	***Queue***
0	- 1 is enqueued - Worker nacks 1 and sleeps for 2-unit time	1 ←head
1	- 2 is enqueued	1 ←head 2 ←tail
2	- Worker awakes - Worker forward 1 to its next destination and acks it	2 ←head
2+delta	- Worker nacks 2 and sleeps for two-time unit	2 ←head
4	- Worker awakes - Worker forward 2 to its next destination and acks it - Worker idle	Empty

V. THE CATLYST: HEADERS

The whole system robustness is driven by the headers present in the record which plays a vital role in enabling the system and to re-play the records again and to delay it. Depending on the delayed times, a same record will have multiple copies at different times as it traverses through multiple topics.

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/25may2034

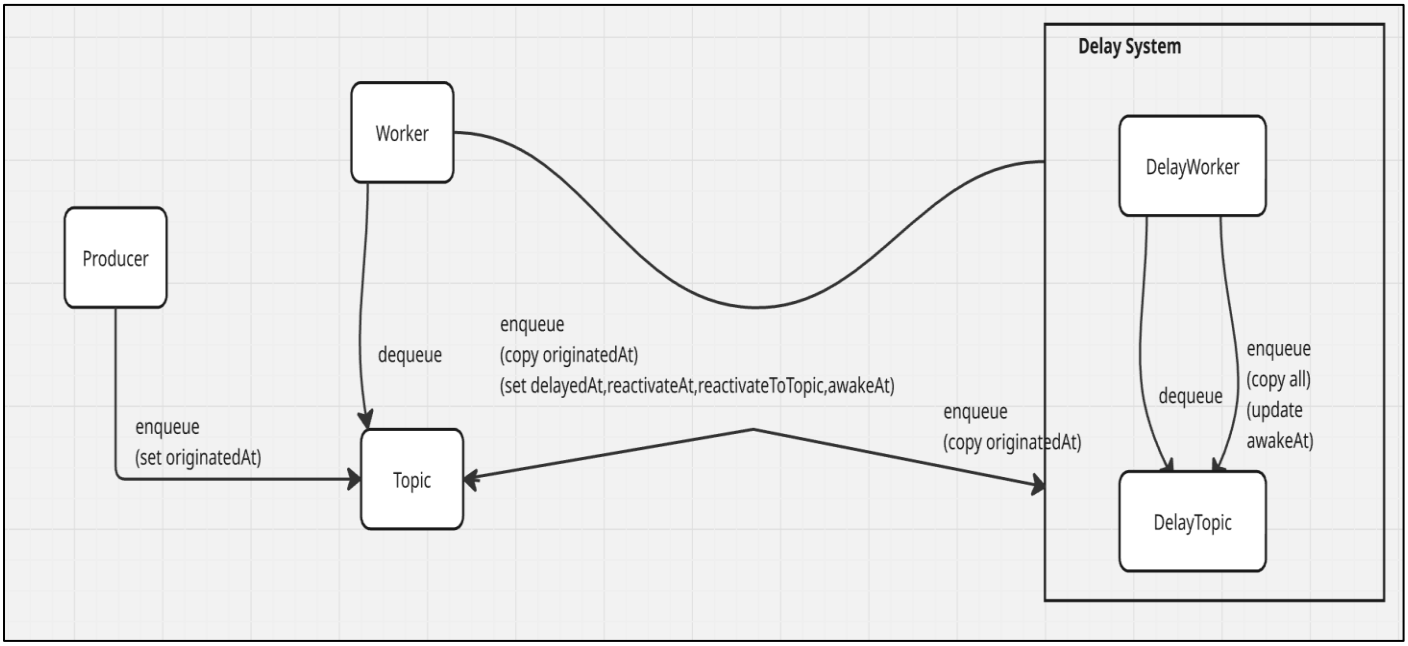

Fig 1 Delay System Header Flow.

A. Irremovable Headers

Some headers persist throughout the lifecycle of the record [7].

- *originatedAt:* The time a record enters the system is used to calculate the total latency.
- *timeRetried:* The replay count for a record helps us set a limit on the number of retries.
- *validAt:* The exact moment a record becomes valid and can be delivered.
- *expireAt:* The moment a record expires, making it ineligible for delivery.

B. B.Removable Headers

Some headers persist only for its lifecycle of the record.

- *delayedAt:* The time a record enters the delay system until it's removed after processing.
- *reactivateAt:* The moment a record can leave the delay system and be added to the reactivateToTopic.
- *reactivateToTopic:* Topic where record should go before existing delay system.
- *awakeAt:* The time a record needs to be checked again to see if it should go to another delay topic or directly to reactivateToTopic.

VI. GOVERNANCE AND RETENTION

The retention of the record in the delay system should be aligned with the Kafka cluster retention policy, which was set during cluster creation. Creating too long retention policy in delay system will not be useful due to Kafka cluster retention limitation. While designing the system different policy can be created which varied in Maximum retires or Number of time unit of delay based on the use case. Before any record to enter the delay system these policies play a critical role as it provides the exit point otherwise record will circulate in infinite loop and will affect system stability.

Records that have exhausted their retry attempts are sent to a dead-letter topic. This provides a dedicated space for review, auditing, or taking corrective actions. The insights gained from these records are also valuable for refining policies and retry schedules, ultimately reducing the number of records that fail in the future

VII. CONCLUSION

This design has demonstrated a robust and innovative approach to implementing a delay subsystem within Apache Kafka. By leveraging Kafka's inherent strengths—its robust write performance, natural sorting capabilities, and reliable data retention—our "denomination" design effectively creates a scalable, resilient, and cost-effective solution for managing delayed messages. This Kafka-native architecture not only eliminates the need for complex external dependencies, thereby simplifying system design and reducing coupling, but also directly addresses critical concerns around data residency, a paramount issue in the banking and fintech sectors. The detailed examination of message flow, worker behavior, and the strategic use of message headers underscores the system's ability to maintain logical record integrity while providing the crucial message scheduling and delaying capabilities often absent in raw Kafka. Ultimately, this approach offers a highly effective and compelling solution for organizations seeking to enhance the asynchronous processing capabilities of their distributed systems without compromising on scalability, reliability, or compliance.

REFERENCES

[1]. Ravi Kiran Mallidi, Manmohan Sharma, Sreenivas Rao Vangala, “Streaming Platform Implementation in Banking and Financial Systems”, 2022,IEEE
[2]. Djerf, Angela, “ A Comparitive Study between EU-GDPR and the US-CCPA” Department of Business Law, 2023, HARN63 20231

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/25may2034

[3]. Abhishek Mhalle, Jianming Yong, Xiaohui Tao, Jun Shen, “Data Privacy and System Security for Banking and Financial Service Industry based on Cloud Computing Infrastructure”,2018,IEEE
[4]. Theofanis P. Rapts, Andrea Passarella, “On Efficieny Partitioning a Topic in Apache Kafka”, 2022, arXiv:2205.09415
[5]. Shreya Gupta, Boyang Huang, Russell Impagliazzo, “The Greedy Coin Change Problem”, 2024, arXiv.2411.18137
[6]. Jiaxin Li,Qian Li, “Analysis of quue management in theme parks introducing the fast pass system”, 2023, Elsevier Ltd
[7]. Dylan Scott, Viktor Gamov, Dave Klein, “Kafka in action”, 2022, Manning