

Original Article

Real-Time Event Processing In Security Camera Systems: An Auto-Sharding and In-Memory Caching Approach Effectively Auto Sharding to Increase Reliability and Cost Efficient

Sibin Thomas

Tech Lead, USA.

Received Date: 19 December 2021

Revised Date: 25 January 2022

Accepted Date: 22 February 2022

Abstract: This paper addresses the challenge of reliable and cost-efficient event processing at scale, a critical requirement for growing companies. Traditional stateless processing approaches often introduce latency and compromise reliability. We propose a solution based on stateful processing and in-memory caching to overcome these limitations. Our approach leverages stateful processing to ensure reliable event handling and reduce latency. Furthermore, by utilizing in-memory caching, we minimize the cost associated with processing large volumes of events. The paper provides a detailed analysis of our proposed solution, demonstrating its effectiveness in achieving both reliability and cost efficiency for large-scale event processing.

Keywords: Security Camera Systems, Event Processing, Auto-Sharding, In-Memory Caching, Real-time Processing, Performance Optimization Cost Efficiency Internet of Things (IoT), Stateful Processing, Optimistic Locking Race Conditions, Latency Reduction.

I. INTRODUCTION

The proliferation of security cameras in modern households has driven a demand for intelligent event detection and processing. These systems typically operate by capturing events, performing preliminary on-device processing, and transmitting relevant data to cloud-based infrastructure. Advanced AI and ML algorithms play a crucial role in accurately identifying and categorizing these events, distinguishing between various sounds (e.g., smoke alarms) and visual occurrences (e.g., human presence, animal movement, familiar faces). This intricate process involves multiple steps, including sophisticated model training on extensive datasets [1], followed by over-the-air deployment to individual cameras [2]. However, the reliance on cloud processing for accurate event identification necessitates the transmission of bursts of sub-events, introducing potential challenges that this paper aims to explore. We will delve into the implications of this architecture, examining its impact on efficiency, latency, and overall system performance.

II. ARCHITECTURE

A. System Architecture of a Typical Home Security Camera

To fully understand the challenges addressed in this paper, a comprehensive overview of a typical home security camera's architecture is necessary. While security cameras involve a complex interplay of components, this analysis will primarily focus on the sound and visual sensors and their associated processes, touching upon video processing as needed. The core function of a home security camera is to provide timely and accurate alerts about events occurring within its field of view (FoV). This process involves multiple steps:

a) Event Detection and Initial Upload:

Upon detecting visual activity, the camera initiates a connection with the camera frontend and transmits a high-level "motion event." Simultaneously, video recording begins to capture the activity.

b) On-Device Event Classification:

To refine the accuracy of the initial motion event, the camera utilizes on-device machine learning models [3]. These models analyze the video frames to categorize the motion as animal, human, or inconsequential (e.g., windblown trees).

c) Further refinement and metadata generation:

If a human is detected, the system employs an on-device human library to compare the detected individual with known faces, further classifying the event as "familiar face" or "unfamiliar face." Throughout this process, the camera also generates metadata events, including timestamps, session identifiers, and track information with pre-roll and post-roll video segments. This metadata ensures chronological organization and contextual awareness for the end-user.



d) *Cloud Processing and Event Consolidation:*

The cloud or device frontend receives this burst of events and metadata within milliseconds. An event channel routes these events to various camera services based on their type. Critically, the events undergo sequential processing and consolidation based on their session ID. This chronological organization is crucial for accurate user notifications, historical data presentation in the user app, and efficient data retention based on user subscription plans.

e) *User Notification and Feed Generation:*

After consolidation and storage in a database [4], relevant events are forwarded to the notification service. This service considers user notification preferences, delivering alerts through the app, or adding events to the user's activity feed.

This complex, multi-step process, while designed for accuracy and comprehensive event capture, inherently involves transmitting a large volume of data to the cloud within a short timeframe. This characteristic introduces challenges that will be explored in subsequent sections.

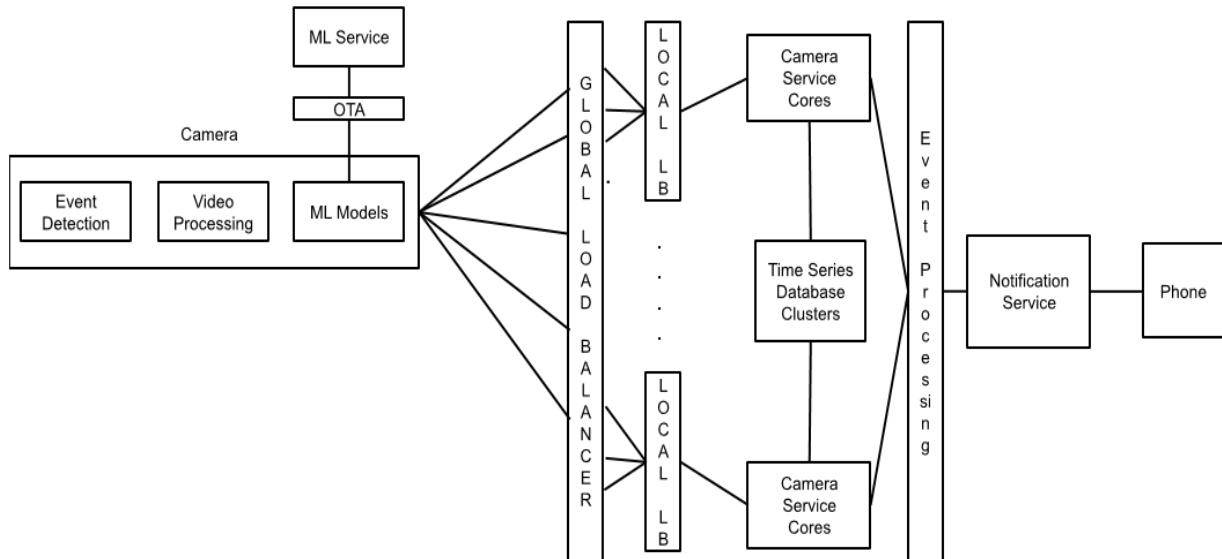


Figure 1: System Architecture

III. PERFORMANCE BOTTLENECKS IN STATELESS EVENT PROCESSING FOR SECURITY CAMERAS

The previously described architecture, while comprehensive in its approach to event capture and organization, suffers from a critical performance bottleneck rooted in its stateless design. This section delves into the specific challenges arising from this statelessness, elucidating their impact on system efficiency and user experience.

At the heart of the issue lies the storage of consolidated session payloads within a time-series database [5]. Each incoming event necessitates retrieving the entire session payload, merging the new event data, and subsequently rewriting the updated payload back to the database. Consider a scenario where a security camera captures a person walking across its field of view. This seemingly simple event can trigger a cascade of sub-events: initial motion detection, human classification, facial recognition attempts, and associated metadata (timestamps, session IDs, etc.). Each of these sub-events, generated within milliseconds, requires a separate read-modify-write cycle for the corresponding session payload in the database.

The stateless nature of cloud processing makes this frequent updating of the same session object worse. Multiple tasks seek to access and modify the same session data at the same time because incoming events are assigned to random processing tasks. This concurrency frequently leads to optimistic lock failures and produces race circumstances [6]. Consider three concurrent jobs handling sub-events for the same session as an example. Every task tries to write the revised payload back to the database after retrieving the session payload and adding the relevant event data. But only one job can write the data and successfully obtain the lock. Optimistic lock failures force the remaining jobs to retry the entire process, which increases database load and contention.

This inherent limitation of the stateless approach translates into three major drawbacks:

A. **Increased Latency:**

Because successful data persistence is required before notification transmission, user notifications are delayed. This latency is increased by optimistic lock failures and retries, which may impede timely user action and real-time event awareness. For example, there may be serious security repercussions if a notification about a possible intruder is delayed.

B. Increased load on camera services:

The processing load on camera services is greatly increased by the retry method. Optimistic lock failures can increase this to 3X or more, assuming an initial event QPS of X. Additionally, failed events are deleted and nacked after a predetermined number of retries (for example, five), necessitating retransmission via the event channel and adding to the processing cost. This effectively means that camera services are unable to handle fresh incoming events since they must spend a significant amount of their resources managing retries and reprocessed events.

C. Increased Load on the Database:

Each event ideally requires two database interactions: one read and one write. However, the combination of optimistic lock failures and increased QPS on camera services can dramatically amplify the database load. In extreme cases, the database can experience a QPS load up to 7X the initial event QPS. This overload strains database resources, potentially leading to performance degradation and impacting the overall system stability. These challenges underscore the limitations of a stateless architecture in managing the high-volume, rapid-fire event streams characteristic of modern security camera systems. The following sections explore potential solutions to overcome these limitations and enhance system performance and efficiency.

IV. MITIGATING PERFORMANCE BOTTLENECKS WITH AUTO-SHARDING AND IN-MEMORY CACHING

To mitigate the performance limitations highlighted in the stateless architecture, we propose a mitigation strategy leveraging auto-sharding and in-memory caching. This approach aims to minimize database contention, reduce latency, and improve overall system efficiency.

A. Auto-Sharding for Deterministic Event Routing:

Deterministic routing of events from a particular device and session to the same processing task is made possible by auto-sharding [7]. We remove the unpredictability present in the stateless method by using both global and local (within area) sharding to guarantee that all events related to a particular session are consistently directed to the same job. This successfully avoids optimistic lock failures and race situations that occur when several jobs compete for the same session data.

B. In-memory Caching for Efficient Data Access

We can further optimize by adding a thread-safe in-memory cache within each task, even while auto-sharding takes care of inter-task contention [8]. In order to perform subsequent events within the same session without requiring a database read, this cache caches recently accessible session data. Database interface and related latency are greatly decreased as a result.

C. Optimized Workflow:

The revised workflow incorporating these optimizations is as follows:

Event Reception and Routing : The camera transmits events to the camera service via the event channel. The event channel, equipped with a sharding-aware client, routes the event to the designated camera service task based on the device and session ID.

a) Initial Cache Miss and Database Interaction:

When the first event of a session arrives at the task, an in-memory cache miss occurs. This triggers a database query to retrieve the session data (if available). The retrieved data is then merged with the incoming event data, and the consolidated payload is stored back in the database. Upon successful database write, the consolidated data, along with the database commit timestamp, is stored in the in-memory cache.

b) Subsequent Cache Hits And Reduced Latency:

Database reads are avoided when subsequent events for the same session result in cache hits. Only the revised payload is written to the database when the new event data and the cached session data are combined. By reducing network communication and removing retries brought on by optimistic lock failures, this method dramatically lowers latency.

c) Periodic Database Updates:

To further optimize performance, we can implement periodic database updates instead of writing to the database for every event. This strategy reduces the frequency of database writes, further lowering latency and minimizing database load.

d) Benefits:

This mitigation strategy offers several key benefits.

e) *Reduced Latency:*

By eliminating database reads for subsequent events within a session and minimizing database writes through periodic updates, we significantly reduce latency, enabling near-real-time event processing and user notification.

f) *Reduced database load:*

In-memory caching dramatically reduces the number of database reads and writes. For instance, if a session generates 10 events, we reduce the database reads from 10 to 1 and potentially reduce writes to just 1. This significantly lowers the database load, improving overall system performance and scalability.

g) *Cost Efficiency:*

Reduced database interactions translate to lower operational costs associated with database usage. By combining auto-sharding and in-memory caching, we address the performance bottlenecks inherent in the stateless architecture, enabling efficient and scalable event processing for modern security camera systems.

V. APPLICATIONS BEYOND SECURITY CAMERAS

The suggested architecture provides a reliable way to handle large event streams and guarantee real-time data availability by combining auto-sharding and in-memory caching. Although it was first created for security camera systems, its fundamental ideas apply to many different applications that share the same requirements: a large volume of data, the need to consolidate data, and the necessity for instant access to data following write operations.

A. Healthcare:

Current healthcare systems produce enormous volumes of patient data, such as test results, medication records, and real-time physiological measurements from monitoring equipment [9]. For prompt diagnosis and treatment, this data must be processed and consolidated effectively. A comprehensive picture of the patient's status can be made possible by auto-sharding, which can guarantee that all data pertaining to a particular patient is consistently routed to the same processing task. Rapid reaction in emergency situations can be facilitated by in-memory caching, which can give instant access to vital signs or recent test results, among other important real-time data.

B. Education:

Educational platforms increasingly rely on data-driven insights to give a personal touch of learning experiences and track student progress [10]. Consider a platform where students engage in interactive exercises, generating a continuous stream of data on their performance and learning patterns. Auto-sharding can ensure that all data for a specific student is processed by the same task, enabling efficient analysis of individual learning trajectories. In-memory caching can facilitate real-time feedback and personalized recommendations by providing immediate access to recent student activity and performance data.

C. Real-time Traffic Management:

As was previously said, smart cities use cameras and vast sensor networks to track traffic patterns and improve traffic control tactics. By directing traffic data from a particular geographic location to the same processing task, auto-sharding can guarantee data localization. By storing frequently used data, such the current traffic conditions on certain routes, in-memory caching can further improve performance by lowering latency and boosting traffic management systems' responsiveness [11].

D. Sports Analytics:

By offering important insights into player performance, game plans, and audience engagement, real-time data analysis is revolutionizing the sports industry [12]. Real-time performance tracking and tactical analysis are made possible by auto-sharding, which can guarantee effective data processing for teams or individual players. Coaches and analysts can make quick decisions by having instant access to vital game statistics thanks to in-memory caching. These illustrations show how adaptable the suggested approach is. We can maximize speed in a variety of data-intensive applications by modifying the fundamental ideas of auto-sharding and in-memory caching. Future studies should look more closely at these applications, examining particular implementation details and adjusting the approach to fit the particulars of each area. This study could lead to novel solutions that improve productivity, lower latency, and advance a number of industries.

VI. CONCLUSION

This paper looks at the problems that come up with stateless event processing in current security camera systems. First, we talked about how these kinds of systems are usually put together, focusing on how complicated it is to find events, sort them into groups, and combine them. The study then looked at the performance problems that come with the stateless method, mainly database contention, optimistic lock failures, and longer wait times. We came up with a way to deal with these problems by mixing auto-sharding and in-memory caching. This method guarantees predictable event routing, cuts

down on database contacts, and makes data available almost in real time. This approach makes the system much more efficient and scalable by lowering latency and database load. Furthermore, we looked into how this system could be used in more areas than just security cameras, applying its ideas to areas like healthcare, education, real-time traffic management, and sports analytics. This research shows how flexible the suggested approach is and how it could improve performance in many different types of data-heavy applications. More study should be done on these applications in the future, making sure that the implementation fits the specifics of each domain. This study can help come up with new ways to improve efficiency, cut down on latency, and ultimately move things forward in many areas. We make it possible for more responsive, scalable, and cost-effective systems that can use the power of real-time data in a world that is becoming more and more linked by solving the problems of processing a lot of events at once.

VI. REFERENCES

- [1] Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep learning. MIT Press.
- [2] Lee, K., & Lee, I. (2015). Over-the-air deployment for embedded systems. *ACM Transactions on Embedded Computing Systems (TECS)*, 14 (3), 1-23.
- [3] Szegedy, Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D.,... & Rabinovich, A. (2015). Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 1-9).
- [4] Chang, F., Dean, J., Ghemawat, S., Hsieh, W., Wallach, D. A., Burrows, M., & Gruber, R. E. (2006). Bigtable: A distributed storage system for structured data. *ACM Transactions on Computer Systems (TOCS)*, 26 (2), 1-26.
- [5] Strauch, C. (2017). *streaming data: Understanding the real-time pipeline*. O'Reilly Media, Inc.
- [6] Kleppmann, M. (2017). *Designing data-intensive applications: The big ideas behind reliable, scalable, and maintainable systems*. O'Reilly Media, Inc.
- [7] Shvachko, K., Kuang, S., Radia, S., & Chansler, R. (2010). the Hadoop Distributed File System. In *2010 IEEE 26th symposium on mass storage systems and technologies (MSST)* (pp. 1-10). IEEE.
- Wang, Zhu, X., Cao, Y., & Liu, J. (2013). Consistent caching with global and local sharding. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data* (pp. 51-62).
- [8] Bates, W., Saria, S., Ohno-Machado, L., Shah, A., & Escobar, G. (2014). Big data in health care: using analytics to identify and manage high-risk and high-cost patients. *Health Affairs*, 33 (7), 1123-1131.
- [9] Siemens & Long, P. (2011). Penetrating the fog: Analytics in learning and educational review, 46(5), 30.
- [10] Zheng, Capra, L., Wolfson, O., & Yang, H. (2014). Urban computing: concepts, methodologies, and applications. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 5(3), 1-55.
- [11] Alamar, C. (2013). *Sports analytics: a guide for coaches, managers, and other decision-makers*. Columbia University Press