The American Journal of Interdisciplinary Innovations and Research
ISSN 2642-7478 Volume 08 - 2026

Implementing Asynchronous Message-Driven Architectures to Achieve Sub-Second Response Time in High-Load Web Applications

Evgenii Lvov

I Head of engineering, senior full stack architect Batumi, Georgia

Received: 31 Jan 2026 | Received Revised Version: 16 Feb 2026 | Accepted: 19 Mar 2026 | Published: 28Apr 2026

Volume 08 Issue 04 2026 | 10.37547/tajiir/Volume08Issue04-06

Abstract

The article presents an analysis of architectural patterns that ensure guaranteed subsecond response time (P99 < 1000 ms) in high-load distributed software systems. The key methodological basis of the study is a comparative analysis of the performance of critical components of asynchronous architecture: message brokers (Apache Kafka, RabbitMQ, Pulsar), interservice communication protocols (REST, gRPC), and concurrency models (Reactive Streams, Virtual Threads), based on the interpretation of empirical data and benchmarks from 2024–2025. The study demonstrates the advantages of binary protocols over text-based ones, establishes the applicability boundaries of different message brokers depending on the load profile, and substantiates the feasibility of using Java 21 virtual threads for handling I/O operations. The results obtained have high practical relevance for systems architects, lead software engineers, and chief technology officers involved in the design of scalable web applications.

Keywords: Asynchronous architecture, microservices, response time, Apache Kafka, gRPC, virtual threads, event-driven architecture, Transactional Outbox, scalability, high-load systems.

Cite This Article: Lvov, E. (2026). Implementing Asynchronous Message-Driven Architectures to Achieve Sub-Second Response Time in High-Load Web Applications. The American Journal of Interdisciplinary Innovations and Research, 8(4), 34–40. https://doi.org/10.37547/tajiir/Volume08Issue04-06

Introduction

In the context of the modern digital economy, the performance of web applications is no longer perceived as an exclusively technical metric and has transformed into a key business parameter determining the market position and resilience of digital platforms. Global markets for e-commerce, fintech services, and cloud solutions operate within the paradigm of the economy of milliseconds, in which request processing latency is in a direct, quantitatively measurable relationship with revenue levels and user retention indicators. Historical and current empirical data demonstrate the fundamental importance of system reaction time: a classical Amazon study showed that an additional delay of only 100 ms leads to an approximately 1% decrease in sales volume [1]. For companies on the scale of the largest retailers, this is comparable to annual losses measured in billions of dollars. Similar patterns are observed by other market participants: Walmart recorded a 2% increase in conversion when page load time was reduced by one second [3]. In the domain of high-frequency trading (HFT), the cost of a unit of latency is even higher: a trading platform lag of 5 ms can result in multimillion losses of forgone profit for a broker [2].

The relevance of the problem under consideration is additionally confirmed by analytical reports from 2024–2025. According to data from Deloitte and Gartner, user expectations regarding service performance continue to tighten, while tolerance for latency is decreasing. In 2024, the average conversion rate in global e-commerce was 1.65%, with a pronounced gap between desktop (3.2%) and mobile (2.8%) platforms, a significant part of which is due to unstable network connections and additional latency arising during content processing on mobile devices [4]. Studies show that about 40% of users leave a website if its loading takes more than three seconds, and 70% of consumers explicitly indicate page performance as a factor influencing their willingness to make a purchase [6]. Under these conditions, achieving sub-second response time, in particular maintaining the 99th percentile of latency (P99) within 1000 ms, becomes an architectural imperative for high-load systems.

The **purpose** of this study is to provide a comprehensive analysis and development of architectural strategies for the transition from synchronous blocking models to asynchronous event-driven architectures (Event-Driven Architecture, EDA).

The scientific novelty is based on the approach proposed in this work: an integrated architectural methodology for achieving guaranteed sub-second response time (P99 < 1000 ms) in high-load web applications through comparative analysis of modern message brokers, RPC protocols, concurrency models, and data consistency patterns, taking into account current benchmarks.

The author's hypothesis is reduced to the assumption that a systematic transition from synchronous blocking models to an asynchronous event-driven architecture using Kafka as the backbone data bus, gRPC with binary serialization, Java 21 virtual threads, and Transactional Outbox/Saga patterns makes it possible to ensure stable sub-second P99 response time in distributed high-load systems without critical degradation of data consistency and without a significant increase in operational costs.

Materials and Methods

To achieve the objective set in the study, a methodological framework of comparative analysis of architectural patterns and technological solutions is employed, based on empirical data extracted from technical reports of leading technology companies (Uber, DoorDash, Netflix), academic publications (IEEE, ACM), as well as independent performance benchmarks.

The following metrics are considered as the basic criteria for evaluating the efficiency of architectural approaches:

– throughput: Quantitatively characterized by the number of messages or requests per second (RPS/TPS). Sufficiently high throughput is a necessary condition for stable system operation under peak loads without noticeable degradation of service quality;

– latency: The focus is placed on the analysis of tail latencies (P95, P99), since average indicators (mean/median) often conceal critical performance issues affecting a significant proportion of users in high-load systems. Within the framework of this study, sub-second response is interpreted as satisfying the condition P99 < 1000 ms;

–resource utilization efficiency: The impact of architectural decisions on CPU and main memory consumption is considered, which is directly related to total cost of ownership (TCO) and energy efficiency.

– Data consistency: The ability of the architecture to ensure data integrity in a distributed environment without critical degradation of availability and performance indicators is evaluated, interpreted within the framework of the CAP theorem.

The analytical part is supplemented by an examination of real industrial migration and optimization cases, which makes it possible to correlate theoretical conclusions with the practical experience of operating web-scale systems and thereby increase the degree of their verifiability.

Results and Discussion

The central element of an asynchronous architecture is the message broker, which provides decoupling of data producers and consumers and thereby minimizes their mutual technological and temporal dependencies. The specific choice of broker effectively shapes the latency and throughput profile of the entire distributed system, since it defines the nature of message exchange, the strategy of data storage and routing, as well as the scalability boundaries. Within the framework of the present study, a detailed comparative analysis was carried out for three dominant solutions in this area:

Apache Kafka, RabbitMQ, and Apache Pulsar.

Apache Kafka is based on a distributed log model in which messages are sequentially written to disk in append-only mode. Such an organization makes it possible to exploit the operating system page cache with maximum efficiency and to achieve very high throughput due to predominantly sequential input/output operations. These architectural decisions are confirmed by benchmark results, according to which Kafka demonstrates the highest throughput compared to alternative solutions [8, 14].

RabbitMQ implements the classical message broker model of the smart broker, dumb consumer type with active use of in-memory indexing. This provides ultra-low delivery latencies (on the order of less than 1 ms) under moderate loads, since key operations of message routing and delivery are performed predominantly in memory, without the need for constant access to disk storage [16]. At the same time, such an architecture exhibits noticeable limitations when transitioning to large-scale deployment scenarios.

Apache Pulsar uses a multi-layer architecture with a clear separation between the compute layer and the storage layer, implemented on the basis of BookKeeper. Such a decomposition increases the flexibility and manageability of scaling in cloud environments, allowing the compute and storage components to evolve independently [5, 14].

The results of the comparative analysis of the peak throughput of the above systems are presented in Figure 1. The obtained data demonstrate a pronounced advantage of Apache Kafka under extreme load regimes, which emphasizes its suitability for high-load data stream processing scenarios.

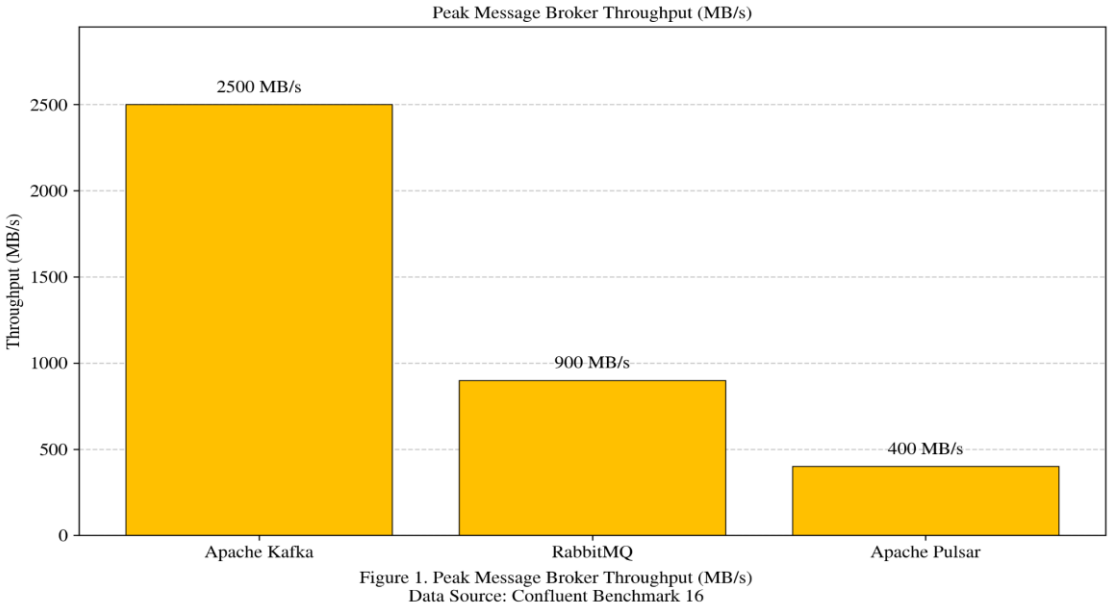

Fig.1. Peak throughput of message brokers (MB/s) (compiled by the author based on [16]).

Analysis of the presented graph shows that Apache Kafka delivers throughput more than six times higher than Apache Pulsar and nearly three times higher than RabbitMQ. This substantial performance margin positions Kafka as a leading choice for building high-throughput data backbones designed to handle continuous, large-scale streaming workloads.

At the same time, in scenarios where the key criterion is minimal latency under relatively moderate load, the use of RabbitMQ may prove to be a more rational choice: its architecture and queue processing model make it possible to optimize precisely message delivery latency rather than the maximum volume of traffic passed through the system.

The consolidated performance characteristics table presented below is formed on the basis of aggregated benchmark results for the period 2022–2024 and reflects the averaged behavior profile of the systems under consideration in typical load scenarios.

Table 1. Comparative characteristics of message brokers (compiled by the author based on [10, 16, 17]).

Characteristic	Apache Kafka	RabbitMQ (Mirrored)	Apache Pulsar
Peak throughput	Very high (~600+ MB/s)	Moderate (~38 MB/s)	High (~300+ MB/s)
P99 latency (load 200 MB/s)	Low (~5 ms)	High (degradation)	Medium (~25 ms)
P99 latency (low load)	Medium (~2 ms)	Ultra-low (~1 ms)	Medium (~5 ms)
Storage model	Log-based (sequential write)	Queue/index-based (in-memory)	Tiered storage (separation of compute/storage)
Primary sub-second scenario	High-throughput stream processing	Low-latency RPC, complex routing	Geo-distributed systems

In asynchronous architectures, not all interaction types can be fully migrated to communication via message brokers; some communication inevitably remains in the form of synchronous calls (RPC), for which the choice of protocol becomes critically important. The widely used REST standard (JSON over HTTP/1.1) is gradually turning into a bottleneck, since the textual JSON format introduces significant overhead for serialization and transmission, and the absence of multiplexing limits the efficiency of network connection utilization.

Migration to gRPC, which relies on Protocol Buffers and the HTTP/2 transport layer, makes it possible to achieve a substantial gain in resource efficiency and latency. Binary serialization in the Protobuf format reduces payload size by approximately 30–50% compared to JSON, which directly affects transfer time and network load. Table 2 below presents a comparison of the key technical characteristics of the protocols under consideration that have a direct impact on performance indicators.

Table 2. Comparison of REST and gRPC protocols (compiled by the author based on [16, 17]).

Characteristic	REST (HTTP/1.1 + JSON)	gRPC (HTTP/2 + Protobuf)	Impact on Sub-Second Response
Data format	Text-based (JSON)	Binary (Protobuf)	Reduces serialization/deserialization time (CPU-bound)
Message size	Large (key redundancy)	Compact	Reduces network load (Network I/O)
Multiplexing	Not supported (Head-of-Line Blocking)	Supported	Enables multiple requests to be executed within a single TCP connection
Typing	Weak (validation required)	Strong (schema .proto)	Eliminates runtime errors and accelerates development

To illustrate the advantages of gRPC under conditions of limited network bandwidth (in particular, when working with mobile clients), Figure 2 shows the comparative distribution of the number of processed requests per second (RPS). This experimental scenario clearly demonstrates how much more efficiently gRPC utilizes the available communication channel compared to alternative protocols while maintaining the required level of quality of service.

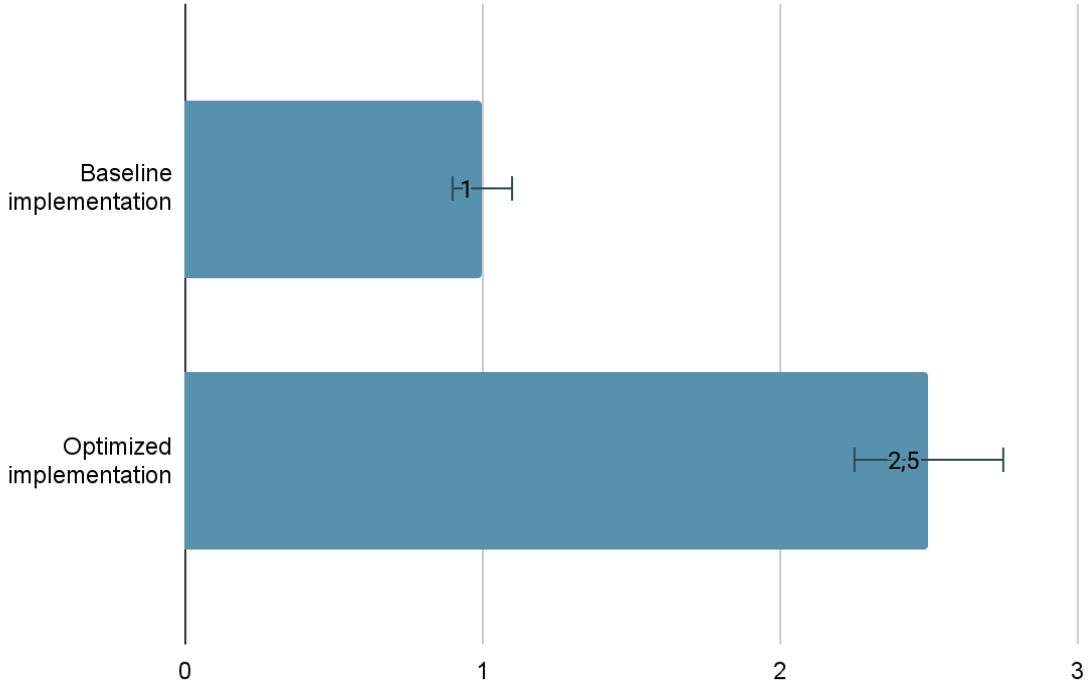

Fig. 2. Comparison of REST vs gRPC performance (Requests Per Second) (compiled by the author based on [15, 17]).

As follows from the data presented in the graph, with identical hardware resources gRPC provides processing of approximately 2.5 times more requests per second. This effect is also confirmed by a practical case from Uber, which migrated the RAMEN push notification platform from SSE to gRPC over QUIC, thereby eliminating transport-level blocking and increasing the reliability of message delivery.

At the code execution level, the struggle for single-digit milliseconds is centered around how efficiently processor threads are utilized. The classical blocking input/output model proves to be fundamentally inefficient under high-load system conditions. For a long time, the standard response to this challenge was the reactive approach (Spring WebFlux); however, the introduction of virtual threads in Java 21 (Project Loom) has radically reshaped the established balance [12, 13].

Virtual threads are lightweight entities whose scheduling and management are performed not by the operating system but by the JVM itself. This makes it possible to use an imperative programming style while achieving efficiency comparable to asynchronous models. The chart demonstrates that the classical model based on platform threads (Platform Threads), as the load increases, exceeds the sub-second response-time regime, reaching approximately 1200 ms, which indicates the occurrence of thread starvation. Virtual threads, by contrast, provide a response time of about 160 ms, that is, at a level practically indistinguishable from the reactive stack (about 150 ms), while significantly reducing the structural and cognitive complexity of the application code.

In asynchronous systems, one of the key problems is the so-called Dual Write Problem, that is, the need to guarantee atomic execution of two operations: persisting data in a relational or other database and sending the corresponding event to a message broker. A

straightforward solution via distributed transactions (2PC) is unacceptable, since it leads to a dramatic drop in performance. Instead, the Transactional Outbox pattern is applied. The client receives a response immediately after the transaction is committed (step 4) and thus is not blocked while waiting for acknowledgment from Kafka. Message delivery (steps 5–6) is performed asynchronously by reading the write-ahead transaction log (WAL), which simultaneously minimizes the latency observed by the user and ensures at-least-once delivery semantics [7, 9].

In large-scale distributed systems comprising hundreds of microservices, local intuitive optimization of individual components, as a rule, does not lead to a noticeable improvement in overall response time. The primary tool for meaningful optimization in this context is critical path analysis (Critical Path Analysis). The CRISP methodology proposed by Uber demonstrates the effectiveness of an approach based on constructing directed acyclic graphs (DAGs) of RPC calls using distributed tracing data (Jaeger). Automated analysis of such graphs makes it possible to identify the longest sequential dependency chain that determines the final request processing latency [11]. Optimization of services that are not part of this critical path does not have a significant effect on the response time perceived by the user.

In addition to architectural patterns, achieving low latency requires targeted optimization of the data storage subsystem. The use of caching solutions for serving hot data sets makes it possible to reduce read time to values on the order of microseconds. A representative example from Uber with the use of an integrated cache on top of Docstore (MySQL) demonstrates the ability to handle millions of requests per second while achieving a P99 latency below 10 ms.

The implementation of sub-second response time is inevitably associated with a conscious trade-off between strong consistency (Strong Consistency) and availability/performance (Eventual Consistency), formalized within the framework of the CAP and PACELC theorems. In scenarios where, minimal latency is critical (for example, a news feed or recommendation systems), the Eventual Consistency model is more preferable. The use of the Saga pattern (in particular, the choreography-based variant) makes it possible to achieve consistency by executing a series of local transactions without resorting to slow global locking protocols (2PC), while maintaining a response time acceptable to the user.

Conclusion

The analysis conducted clearly demonstrates that achieving sub-second response time in high-load web applications is possible only through a systemic transition from synchronous blocking architectures to asynchronous event-driven design models.

The comparison results confirm that Apache Kafka is the most rational choice as the backbone data bus for large-scale systems: under peak load its throughput can exceed that of RabbitMQ by approximately 15 times. At the level of inter-service communication, abandoning REST in favor of gRPC with binary serialization based on Protobuf reduces network and serialization overhead, which results in latency reductions of 50–70% and an increase in throughput by approximately 2.5 times.

In the context of runtimes, Java 21 virtual threads demonstrate the potential to become the de facto standard for I/O-intensive scenarios, providing P99 latencies on the order of ~160 ms, comparable to reactive frameworks, while imposing significantly lower cognitive and architectural complexity on developers. The problem of ensuring data integrity in such asynchronous systems is effectively solved through the application of the Transactional Outbox pattern in combination with CDC tools, which makes it possible to strongly decouple database write operations from event publication without sacrificing reliability and delivery guarantees.

In the conditions of an economy of milliseconds, investments in asynchronous architecture go far beyond local technical optimization and become a strategic factor of competitiveness, exerting a direct impact on the company's financial performance and the level of user satisfaction.

References

1. Latency Reduction – The Competitive Edge in Modern Markets. Retrieved from: https://wealthandfinance.digital/latency-reduction-the-competitive-edge-in-modern-markets/ (date accessed: November 7, 2025).

2. The Cost of Latency | Digital Realty. Retrieved from: https://www.digitalrealty.co.uk/resources/articles/th

e-cost-of-latency (date accessed: November 8, 2025).

3. 20+ Interesting Website Speed Statistics (2025). Retrieved from: https://www.sitebuilderreport.com/website-speed-statistics (date accessed: November 8, 2025).

4. 50 E-commerce Conversion Rate Statistics for 2025 . Retrieved from: https://www.envive.ai/post/ecommerce-conversion-rate-statistics (date accessed: November 9, 2025).

5. E-commerce conversion rate benchmarks - 2025 update . Retrieved from: https://www.smartinsights.com/ecommerce/ecommerce-analytics/ecommerce-conversion-rates/ (date accessed: November 10, 2025).

6. Famoti, O., Omowole, B. M., Nzeako, G., Shittu, R. A., Ezechi, O. N., Ewim, C. P. M., & Omokhoa, H. E. (2025). A digital transformation framework for US e-commerce supply chains. International Journal of Scientific Research in Computer Science, Engineering and Information Technology. 2025d, 11(1), 1670-1701.

7. What is Synchronous and Asynchronous Programming: Differences & Guide . Retrieved from: https://kissflow.com/application-development/asynchronous-vs-synchronous-programming/ (date accessed: November 10, 2025).

8. Spring MVC vs. Spring WebFlux: Choosing the Right Framework for Your Project . Retrieved from: https://dev.to/jottyjohn/spring-mvc-vs-spring-webflux-choosing-the-right-framework-for-your-project-4cd2 (date accessed: November 10, 2025).

9. El Akhdar, A., Baidada, C., Kartit, A., Hanine, M., García, C. O., Lara, R. G., & Ashraf, I. (2024). Exploring the Potential of Microservices in Internet of Things: A Systematic Review of Security and Prospects. Sensors, 24(20), 6771. https://doi.org/10.3390/s24206771

10. Padmanaban, K., Babu, T. G., Karthika, K., Pattanaik, B., & Srinivasan, C. (2024, October). Apache Kafka on Big Data Event Streaming for Enhanced Data Flows. In 2024 8th International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC) (pp. 977-983). IEEE. https://doi.org/10.1109/I-SMAC61858.2024.10714884.

11. Wang, T., & Qi, G. (2024). A comprehensive survey on root cause analysis in (Micro) services: methodologies, challenges, and trends. arXiv preprint arXiv:2408.00803. https://doi.org/10.48550/arXiv.2408.00803.

12. Iurchenko, A. (2025). Optimization of Microservices Architecture Performance in High-Load Systems. The American Journal of Engineering and Technology, 7(05), 123-132. https://doi.org/10.37547/tajet/Volume07Issue05-10.

13. Spring Boot: MVC vs WebFlux - Sharkbench. Retrieved from: https://sharkbench.dev/web/java-springboot/mvc-vs-webflux (date accessed: November 10, 2025).

14. Chy, M. S. H., Arju, M. A. R., Tella, S. M., & Cerny, T. (2023). Comparative Evaluation of Java Virtual Machine-Based Message Queue Services: A Study on Kafka, Artemis, Pulsar, and RocketMQ. Electronics, 12(23), 4792. https://doi.org/10.3390/electronics12234792.

15. Vyas, S., Tyagi, R. K., Jain, C., & Sahu, S. (2021, July). Literature review: A comparative study of real time streaming technologies and apache kafka. In 2021 Fourth International Conference on Computational Intelligence and Communication Technologies (CCICT) (pp. 146-153). IEEE. https://doi.org/10.1109/CCICT53244.2021.00038.

16. Putra, S. J., Firmansyah, G., Tjahjono, B., & Akbar, H. (2025). Comprehensive Benchmarking of Message Brokers: Evaluating Performance and Security Metrics for Reliable Messaging Systems. Jurnal Locus Penelitian dan Pengabdian, 4(11), 10829-10838.https://doi.org/10.58344/locus.v4i11.5081.

17. Odofin, O. T., Owoade, S., Ogbuefi, E., Ogeawuchi, J. C., & Segun, O. (2022). Integrating Event-Driven Architecture in Fintech Operations Using Apache Kafka and RabbitMQ Systems. Int. J. Multidiscip. Res. Growth Eval, 3(4), 635-643. https://doi.org/10.54660/.IJMRGE.2022.3.4.635-643.