UDC 004.75:37.018.43
DOI https://doi.org/10.24144/2616-7700.2026.49(2).173-180

T. Hombosh[1], P. Mulesa[2]

[1] Uzhhorod National University,
PhD student of the Department of Cybernetics and Applied Mathematics
tomash.hombosh@uzhnu.edu.ua
ORCID: https://orcid.org/0009-0006-6282-7689

[2] Uzhhorod National University,
Head of the Department of Cybernetics and Applied Mathematics,
Doctor of Pedagogical Sciences, Associate Professor
pavlo.mulesa@uzhnu.edu.ua
ORCID: https://orcid.org/0000-0002-3437-8082

STREAM EVENT PROCESSING PATTERNS FOR SCALING INTERACTIVE LEARNING ENVIRONMENTS: A COMPARATIVE ANALYSIS OF REDIS PUB/SUB, REDIS STREAMS, AND APACHE KAFKA

This article explores how to best structure real-time stream event processing to handle the increasing demands of interactive learning environments. We've noticed that as more and more users jump into collaborative coding and interactive whiteboard sessions, traditional methods like Redis Pub/Sub are starting to show their limits. The problem? Packet loss and events arriving out of order. This is a big deal when you're trying to keep things synchronized using algorithms like Conflict-free Replicated Data Types (CRDT) and Operational Transformation (OT). So, we ran a comparison of three message broker technologies—Redis Pub/Sub, Redis Streams, and Apache Kafka—in a microservice setup using NestJS and Socket.IO. We also came up with a way to categorize interactive learning events into three groups based on how often they happen and how reliable and ordered they need to be. Finally, we put together six key criteria for picking the right message broker for educational situations where there's a lot of back-and-forth. This forms the core of our research. An architectural pattern for Event Sourcing, leveraging Redis Streams as the definitive source for real-time collaborative sessions, is presented alongside a client reconnection strategy. Its practical value lies in delivering an accessible and easily replicated toolkit, integrated with Learning Management Systems, that guarantees robust synchronization, even with the fluctuating network conditions typical in student environments.

Keywords: interactive learning, event-driven architecture, Redis Streams, Apache Kafka, CRDT, microservices, Socket.IO, real-time synchronization.

1. Introduction. The modern educational paradigm is increasingly shifting towards active, interactive learning, where students participate directly in their own learning process. The development of specialized tools for interactive learning – collaborative code editors, shared whiteboards, and gamified assessment systems – requires software architectures capable of processing thousands of events per second while maintaining strict consistency guarantees [8].

Previous research established the transition from monolithic solutions to microservice architecture using the NestJS, Socket.IO, and Redis stack for basic real-time interaction, as well as the implementation of CRDT and OT algorithms for state coordination [1]. However, as the system scales – when the number of students in a shared workspace grows during collaborative activities such as joint code editing or interactive drawing – the basic Publish/Subscribe pattern based on Redis Pub/Sub exhibits significant limitations. The absence of persistence leads to packet

loss During brief connection interruptions, and the lack of ordering guarantees destroys the consistency of CRDT structures, causing irreversible state divergence across replicas.

The **goal** of this article is to develop criteria for selecting and applying message broker architectural patterns for scaling interactive learning environments with high-frequency events.

The **objectives** are: (1) to classify event types by frequency, reliability, and ordering requirements; (2) to conduct a comparative analysis of Redis Pub/Sub, Redis Streams, and Apache Kafka within a NestJS and Socket.IO microservices context; (3) to formulate broker selection criteria for educational high-frequency interaction scenarios; (4) to propose architectural patterns for guaranteed delivery and causal ordering of events to ensure CRDT consistency.

2. Review of Related Work. The problem of scaling WebSocket connections and state synchronization in distributed systems has been actively studied in the context of IoT and financial technologies. However, educational platforms (LMS) have a specific load profile: pronounced session-based patterns (high load during lectures, near-zero at night), a high frequency of small events (cursor movements, individual keystrokes), and the critical importance of causal event ordering for collaborative work. Kleppmann et al. [3] demonstrated that even a theoretically correct CRDT implementation diverges if the event delivery mechanism does not guarantee causal ordering. Almeida [4] surveys CRDT families and highlights that operation-based CRDTs require exactly-once, causally ordered delivery – a requirement Redis Pub/Sub cannot satisfy in multi-node deployments. David et al. [5] showed that under 1% packet loss, Pub/Sub systems experienced divergence in over 12% of collaborative sessions. Srivastava et al. [6] confirmed that Redis, Kafka, and RabbitMQ occupy distinct niches in the latency-throughput-reliability trade-off. Kreps et al. [7] established the append-only log model underlying both Kafka and Redis Streams. Newman [8] provides the microservices and EDA framework underpinning the proposed solution. The gap in existing research is the absence of broker selection criteria adapted to educational platforms, addressed in Section 5.

3. Classification of Events in Interactive Learning Environments. We propose a three-tier classification of LMS events (Table 1).

The three-tier model provides the foundation for the selection criteria formulated in Section 6.

4. Comparative Analysis of Message Brokers.

4.1. Redis Pub/Sub. Used as a “fire-and-forget” adapter between WebSocket server instances. *Advantages:* sub-millisecond latency, zero configuration. *Limitations:* no persistence, no replay, no ordering guarantee across nodes. *Verdict:* suitable only for *Ephemeral Presence* events.

4.2. Redis Streams. An append-only log with Consumer Groups (Redis 5.0) [9]. *Advantages:* in-memory persistence, replay via `XRANGE`, strict ordering within a single stream, 1–3 ms latency. *Ordering caveat:* in sharded or clustered Redis deployments, cross-stream ordering is not guaranteed; Consumer Group redelivery on failure may cause out-of-order processing, requiring application-level sequence validation [9]. *Limitations:* RAM-bounded; requires export for long-term retention. *Verdict:* optimal for *Collaborative Editing (CRDT/OT).*

4.3. Apache Kafka. A distributed log for high-throughput durable storage [7].

Table 1.

Classification of events in interactive learning environments

Event Type	**Examples**	**Frequency (per user)**	**Ordering Requirement**	**Loss Tolerance**
Ephemeral presence	Cursor movement, "typing..." status, online/offline indicator	Very high (10–30 ev/s)	Low (only the latest state matters)	High (intermediate frames non-critical)
Collaborative editing (CRDT/OT)	Character insertion in code, vector addition on whiteboard, shape resize	Medium (1–5 ev/s)	**Critical** (strict causal order required)	**None** (causes irreversible state divergence)
Analytics and assessment	Test answer submission, page navigation, session start/end	Low (<1 ev/s)	Medium (chronological)	None (affects student grade and audit trail)

Exactly-once semantics are achievable only under specific conditions: `enable.idempotence=true`, transactional producer APIs, and coordinated consumer configuration; this guarantee is scoped to individual partitions and does not extend to side effects outside the broker. *Advantages:* guaranteed persistence, horizontal scaling, terabyte-scale analytics. *Limitations:* 10–50 ms latency, high operational complexity (ZooKeeper/KRaft). *Verdict:* optimal for *Analytics and Assessment* and asynchronous LLM hint generation [2].

5. Architectural Patterns for Guaranteed Delivery and Causal Ordering.

5.1. Event Sourcing with Redis Streams as Single Source of Truth. The pipeline proceeds as follows: (1) a client generates a CRDT operation and sends it via Socket.IO to the nearest NestJS Gateway; (2) the Gateway writes it to a room-specific Redis Stream (`room:123:events`) via `XADD` — not directly to clients; (3) Redis assigns a monotonically increasing ID, fixing global order; (4) all Gateway instances consume the stream via `XREAD` and forward events to clients via WebSocket. This decouples production from consumption and ensures all replicas process events in the same monotonic order within a single non-sharded stream — a necessary precondition for CRDT convergence. Applications requiring cross-stream causal ordering must additionally implement vector clocks or Lamport timestamps at the application layer.

5.2. Catch-Up Mechanism for Reconnecting Clients. Each client stores the last received event ID. On reconnection, the server queries `XRANGE room:123:events $lastId +` and returns all missed events in order. The client applies them sequentially — orders of magnitude less data than a full state snapshot for a typical 1000-operation session.

5.3. NestJS Implementation. Listing 1 illustrates the core Gateway logic for writing a collaborative editing event to a Redis Stream using the `ioredis` library

Table 2.

Comparative characteristics of message brokers for interactive learning

Criterion	**Redis Pub/Sub**	**Redis Streams**	**Apache Kafka**
Latency	<1 ms (best)	1–3 ms (excellent)	10–50 ms (acceptable)
Delivery guarantee	None (at-most-once)	At-least-once; Consumer Groups may redeliver on failure	Exactly-once within a partition under idempotent producer and transactional API configuration only
Event ordering	Not guaranteed across nodes	Strict within a single stream; not guaranteed across shards or in Consumer Group redelivery scenarios	Strict within a partition; requires idempotent producer configuration
Persistence	None	Temporary (session)	Long-term (years)
Replay on reconnect	Not supported	Supported (`XRANGE`)	Supported (offset reset)
Target LMS scenario	Cursor broadcast, mic status	Collaborative editing, CRDT	Test results, LLM analytics
Infrastructure cost	Low	Medium (RAM-bound)	High (Disk I/O, CPU)

in a NestJS service.

5.4. Hybrid Broker Architecture. Based on the event classification in Section 3 and the broker analysis in Section 4, we propose a hybrid architecture that assigns each event category to the most appropriate broker (Fig. 2).

6. Selection Criteria for Message Brokers in Educational Scenarios. Existing broker comparison frameworks [6, 8] address general distributed systems without accounting for the specific constraints of educational platforms: session-based bursty load, tolerance for ephemeral event loss, and the causal ordering requirements of CRDT algorithms. The following six criteria are formulated specifically for this context, constituting the **scientific novelty** of the study.

Criterion 1: Latency threshold. Latency must not exceed 50–100 ms for synchronous activities [3]. Redis Pub/Sub and Streams satisfy this unconditionally; Kafka requires tuning.

Criterion 2: Ordering guarantee scope. CRDT convergence requires causal ordering within a session. Redis Streams guarantees this within a stream; Kafka within a partition; Pub/Sub does not.

Criterion 3: Reconnection support. Transient disconnections require catch-up replay. Redis Streams and Kafka support this natively; Pub/Sub does not.

Criterion 4: Session-based load profile. Educational load is bursty. Redis

```
@WebSocketGateway()
@Injectable()
export class CollabGateway {
  constructor(private readonly redis: Redis) {}

  @SubscribeMessage('crdt:operation')
  async handleCrdtOp(
    @ConnectedSocket() client: Socket,
    @MessageBody() payload: CrdtOperationDto,
  ): Promise<void> {
    const streamKey = `room:${payload.roomId}:events`;
    const eventId = await this.redis.xadd(
      streamKey, '*', 'type', payload.type,
      'clientId', client.id, 'data',
      JSON.stringify(payload.data),
      'clock', payload.vectorClock,
    );
    await this.redis.xtrim(
      streamKey, 'MAXLEN', '~', 10000
    );
  }
}
```

Figure 1. NestJS Gateway: writing a CRDT operation to Redis Streams.

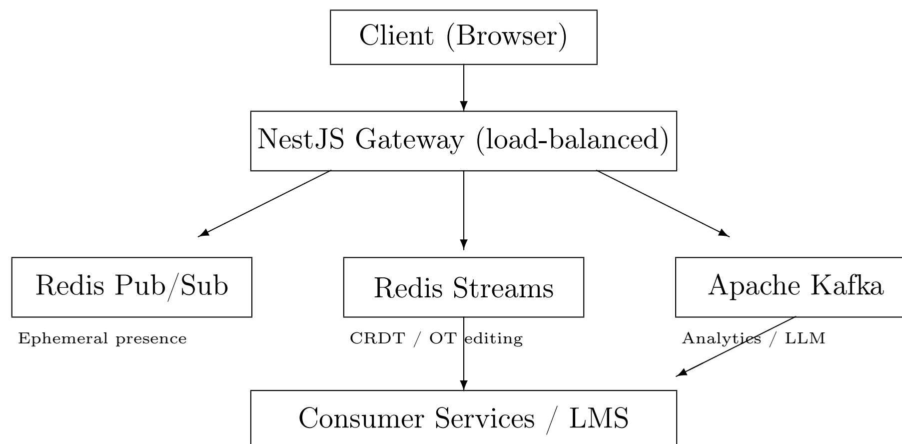

Figure 2. Hybrid message broker architecture for interactive learning environments.

Streams scales with RAM; Kafka requires a pre-provisioned dedicated cluster.

Criterion 5: Operational complexity budget. Redis Streams deploys within an existing Redis instance; Kafka requires ZooKeeper/KRaft, monitoring, and schema registry.

Criterion 6: Data retention requirements. Grades require 1–5 years re-

tention. Kafka provides this natively; Redis Streams requires periodic export to PostgreSQL or S3.

Applying these criteria to the three-tier event classification yields the broker assignment summarised in Table 3.

Table 3.

Broker selection matrix for interactive learning event categories

Event Category	**Recommended Broker**	**Decisive Criteria**
Ephemeral presence	Redis Pub/Sub	Criterion 1 (latency <1 ms); loss tolerance is high; no ordering needed
Collaborative editing (CRDT/OT)	Redis Streams	Criteria 1, 2, 3 (low latency + strict ordering + reconnection catch-up)
Analytics and assessment	Apache Kafka	Criteria 6, 2 (long-term retention + partition-level ordering + exactly-once under transactional configuration)

7. Experimental Modelling and Expected Results. The test environment consists of three NestJS instances, an Nginx load balancer, a Redis cluster, and a Kafka cluster in Docker Compose (32 GB RAM, 8 CPU cores), simulating 1000 concurrent students across 50 rooms at 5 events/s each (5000 events/s total) with 1% packet drop and 20 ms jitter via `tc netem`.

The end-to-end latency T for a CRDT operation is modelled as:

$$T = T_{\text{client}} + T_{\text{network}} + T_{\text{gateway}} + T_{\text{broker}} + T_{\text{fan-out}}, \quad (1)$$

where T_{client} is the client-side serialisation time, T_{network} is the round-trip network time, T_{gateway} is the NestJS processing time, T_{broker} is the broker write and read time, and $T_{\text{fan-out}}$ is the time to broadcast the confirmed event to all room participants via Socket.IO.

The synchronisation reliability metric R is defined as the fraction of collaborative sessions that maintain full CRDT consistency (zero divergence) over a 30-minute session:

$$R = 1 - \frac{N_{\text{diverged}}}{N_{\text{total}}}, \quad (2)$$

where N_{diverged} is the number of rooms where at least one replica diverged from the ground-truth state, and N_{total} is the total number of rooms.

The evaluation methodology explicitly controls for three dimensions identified as critical in comparative broker studies [6]: (1) *system configuration* — each broker is tested under both default and latency-optimised settings (`linger.ms=0`, `acks=1` for Kafka; `appendfsync=everysec` for Redis); (2) *load distribution* — load is applied both uniformly across rooms and unevenly (80% of events concentrated in 20% of rooms) to simulate lecture-peak conditions; (3) *failure behaviour* — one Gateway instance is forcibly terminated mid-session to measure recovery time and event loss under each broker.

Expected results:

- *Redis Pub/Sub:* Up to 2–5% event loss under 1% network packet drop, leading to irreversible CRDT divergence in approximately 15% of rooms ($R \approx 0.85$). $T_{\text{broker}} < 1$ ms.
- *Redis Streams:* 0% event loss due to the catch-up mechanism. $T_{\text{broker}} \approx 2$–3 ms. Total end-to-end latency expected at 15–25 ms, imperceptible to users. $R = 1.0$.
- *Apache Kafka:* 0% event loss with exactly-once semantics. $T_{\text{broker}} \approx 15$–50 ms. Total latency may reach 50–80 ms, acceptable for analytics but suboptimal for real-time drawing. $R = 1.0$.

8. Conclusions and Prospects for Further Research. The following results have been obtained: (1) a three-tier event classification (ephemeral presence, collaborative editing, analytics) providing a principled basis for broker selection; (2) a comparative analysis showing no single broker is optimal for all LMS event categories; (3) six broker selection criteria for educational high frequency scenarios, constituting the scientific novelty; (4) an Event Sourcing pattern with Redis Streams as single source of truth and a catch-up mechanism for reconnecting clients; (5) a hybrid broker architecture assigning each event category to the most appropriate broker.

Prospects include empirical validation under realistic load, vector clock-based causal ordering at the application layer, and LLM-based real-time pedagogical feedback integration.

Conflict of Interest

The authors declare no conflicts of interest.

Funding

The research was conducted without financial support.

Data Availability

All data are available in the main text.

Use of artificial intelligence

The authors confirm that they did not use artificial intelligence technologies when creating the current work.

Contributions of authors

Hombosh T.: Conceptualization, Methodology, Software, Writing — original draft. Mulesa P.: Supervision, Validation, Writing — review & editing.

Copyright © (2026). Hombosh T., Mulesa P. This work is licensed under a Creative Commons Attribution 4.0 International License.

References

1. Hombosh, T., & Mulesa, P. (2025). Securing Interactive Learning Tools: CRDT/OT, Signed

Events, and Analytics. In *Proceedings of the Scientific and Methodological Conference on Informatics, Computer Science and Mathematics (SMICS-2025).* Lviv: Ivan Franko National University of Lviv, 356. Retrieved from https://smics.lnu.edu.ua/uk/zbirnyk/
2. Hombosh, T., & Mulesa, P. (2025). Automated Llama-based generation of school test items: alignment and localization to Ukrainian educational standards. In *Abstracts of the International Scientific Conference dedicated to Academician M. I. Kravchuk.* Kyiv: Igor Sikorsky Kyiv Polytechnic Institute, 180. Retrieved from https://matan.kpi.ua/media/2025/kravchuk-conf-2025/kravchuk-conf-2025-abstracts.pdf
3. Kleppmann, M., Gentle, J., Litt, G.,& Litt, A. (2022). Peritext: A CRDT for Collaborative Rich Text Editing. *Proceedings of the ACM on Human-Computer Interaction,* 6(CSCW2), 1–36. https://doi.org/10.1145/3555644
4. Almeida, P. S. (2023). Approaches to Conflict-free Replicated Data Types. *arXiv preprint arXiv:2310.18220.* Retrieved from https://arxiv.org/abs/2310.18220
5. David, I., Syriani, E., & Masson, C. (2022). Extensible Conflict-Free Replicated Datatypes for Real-time Collaborative Software Engineering. In *Proceedings of the 2022 Annual Conference on Computer Science and Information Systems (FedCSIS 2022),* 99–108. Retrieved from https://annals-csis.org/proceedings/2022/drp/pdf/99.pdf
6. Srivastava, A., et al. (2025). Comparing Big Data Messaging Platforms: An Evaluation of Apache Kafka, RabbitMQ, and Redis. In *Proceedings of the IEEE 16th Annual Ubiquitous Computing, Electronics & Mobile Communication Conference (UEMCON 2025).* https://doi.org/10.1109/UEMCON67449.2025.11267649
7. Kreps, J., Narkhede, N., & Rao, J. (2011). Kafka: A Distributed Messaging System for Log Processing. In *Proceedings of the 6th International Workshop on Networking Meets Databases (NetDB 2011).* Athens, Greece. Retrieved from https://www.microsoft.com/en-us/research/wp-content/uploads/2017/09/Kafka.pdf
8. Newman, S. (2021). *Building Microservices: Designing Fine-Grained Systems* (2nd ed.). Sebastopol, CA: O'Reilly Media.
9. Redis Ltd. (2024). *Redis Streams: Introduction to Redis Streams.* Retrieved from https://redis.io/docs/latest/develop/data-types/streams

Гомбош Т., Мулеса П. Патерни обробки потокових подій для масштабування інтерактивних навчальних середовищ: порівняльний аналіз Redis Pub/Sub, Redis Streams та Apache Kafka.

У статті досліджуються архітектурні патерни обробки потокових подій у реальному часі для масштабування інтерактивних навчальних середовищ. Зростання кількості одночасних користувачів у сценаріях спільного редагування коду та використання інтерактивних дошок виявляє обмеження традиційних підходів на базі Redis Pub/Sub, зокрема щодо втрати пакетів та порушення порядку доставки подій, що є критичним для алгоритмів узгодження стану (CRDT та OT). Проведено порівняльний аналіз трьох технологій брокерів повідомлень — Redis Pub/Sub, Redis Streams та Apache Kafka — в контексті мікросервісної архітектури на базі NestJS та Socket.IO. Запропоновано трирівневу класифікацію подій інтерактивного навчання за частотою, надійністю та вимогами до впорядкування. Сформульовано шість критеріїв вибору брокера повідомлень специфічно для навчальних сценаріїв з високою астотою подій, що становить наукову новизну дослідження. Описано архітектурний патерн Event Sourcing з використанням Redis Streams як єдиного джерела істини для активних сесій спільної роботи, а також механізм наздоганяння (catch-up) для клієнтів, що перепідключаються. Практичне значення полягає у можливості створення відкритого відтворюваного інструментарію, інтегрованого з LMS, що забезпечує високу надійність синхронізації навіть за нестабільного мережевого з'єднання.

Ключові слова: інтерактивне навчання, подійно-орієнтована архітектура, Redis Streams, Apache Kafka, CRDT, мікросервіси, Socket.IO, синхронізація в реальному часі.

Recived: 05.04.2026 Accepted: 21.04.2026 Published: 30.04.2026