Buletin Ilmiah Sarjana Teknik Elektro
Vol. 8, No. 2, April 2026, pp. 576-584
ISSN: 2685-9572, DOI: 10.12928/biste.v8i2.14214

Optimizing Event Replay with Hybrid Snapshotting in Event-Sourced Systems

Akhulbay Shyngys [1], Azamat Serek [2,3], Darkhan Kuanyshbay [4]
[1] School of Information Technologies and Engineering, Kazakh-British Technical University, Almaty, Kazakhstan
[2] Astana IT University, Astana, Kazakhstan
[3] School of Digital Technology, Narxoz University, Almaty, Kazakhstan
[4] Department of Information Systems, SDU University, Kaskelen, Kazakhstan

ARTICLE INFORMATION

Article History:

Received 08 July 2025
Revised 28 November 2025
Accepted 08 May 2026

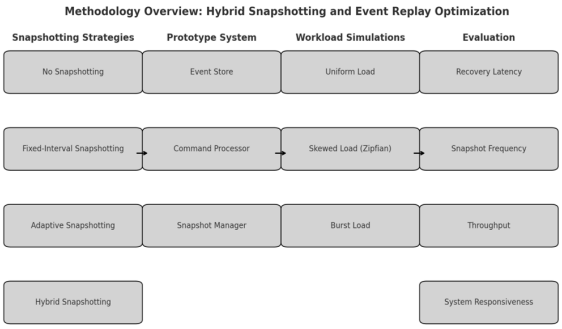
Keywords:

Event Sourcing;
State Reconstruction Latency;
Hybrid Snapshotting;
Distributed Event Stores;
Recovery Time Optimization

Corresponding Author:

Azamat Serek,
Astana IT University,
Astana, Kazakhstan.
Email:
Azamat.Serek@astanait.edu.kz

ABSTRACT



This study presents a hybrid framework for snapshotting and event replay optimization aimed at improving performance and scalability in event-sourced systems, where state reconstruction frequently experiences significant latency due to the replay of extensive event streams. Four snapshotting strategies were used to test this method: No Snapshotting, Fixed-Interval, Adaptive, and the new Hybrid method. The Adaptive strategy used dynamic triggers to make snapshots when an aggregate had more than 1,000 accesses in a five-minute window, when more than fifteen minutes had passed since the last snapshot, or when its event stream had more than 5,000 events. The Hybrid method used these adaptive triggers along with a fixed base interval of 5,000 events. For hot aggregates, the interval was cut in half to balance systematic coverage with runtime responsiveness. We built a prototype system in Java with Spring Boot and PostgreSQL. We used Kryo for snapshot serialization and Kafka to make controlled workloads. We ran tests on a 16-core Intel Xeon server to see how well it could handle event distributions that were uniform, skewed (Zipfian), or bursty, with anywhere from 10,000 to 1,000,000 events. We ran each configuration ten times and used 95% confidence intervals to find the average of the results. The results show that the Hybrid strategy consistently cut recovery latency, getting it down to 40 seconds under burst loads. This is a 58% improvement over Fixed-Interval and a 38% improvement over Adaptive. Throughput and responsiveness during normal operation were not affected, which is important. These results show that Hybrid snapshotting strikes a good balance between speed of recovery and storage overhead. This makes it a good choice for real-time, event-sourced applications. In the future, we will look into automatic tuning of snapshot intervals and adaptive thresholds, as well as adding support for distributed deployments and complex event schemas.

Document Citation:

A. Shyngys, A. Serek, and D. Kuanyshbay "Optimizing Event Replay with Hybrid Snapshotting in Event-Sourced Systems" *Buletin Ilmiah Sarjana Teknik Elektro*, vol. 8, no. 2, pp. 576-584, 2026, DOI: 10.12928/biste.v8i2.14214.

1. INTRODUCTION

In the age of distributed, highly available, and fault-tolerant systems, event sourcing has become a key architectural pattern for making sure that data is consistent, can be traced, and that systems are resilient [1]. Event sourcing records every change to an application's state as a series of unchangeable events. This is different from traditional CRUD-based persistence models. This method makes it possible to fully audit, makes it easier to run time-based queries, and makes debugging easier by letting you rebuild the system state at any time [2]. As systems grow and the number of stored events grows, event-sourced architectures' performance and scalability become more and more of a problem, especially when it comes to state reconstruction through event replay [3].

Despite its advantages, prior studies have primarily focused on general performance optimization or adjacent domains, leaving a clear research gap: existing snapshotting techniques for event-sourced systems often struggle to adapt to diverse workload patterns without incurring excessive storage overhead or recovery latency. Given the rise of AI [4]-[6], previous references to areas like natural language processing [7][8] and fraud detection [9] offer helpful comparisons for working with sequential data, but they don't directly address the specific needs of event replay and snapshotting. This lack of connection shows that there needs to be a focused study of snapshotting strategies that are specifically designed for event-sourced architectures.

One of the main problems with large-scale event-sourced systems is that it costs a lot to replay long event streams to get the current state of aggregates [10][11]. Replaying tens or hundreds of thousands of events for a single entity can cause a lot of lag when an application starts up, recovers from a crash, or moves from one service to another [12]. This latency can make systems less available and responsive, especially in environments where latency is low, like real-time financial systems or IoT applications [13]. To fix this, snapshotting techniques are often used. These techniques take a picture of an entity's current state and store it next to the event log on a regular basis [14]. After recovery, the system can restore the most recent snapshot and only play back the events that happened after that, which cuts down on the time it takes to rebuild the current state [15].

Traditional snapshotting methods, like fixed-interval snapshotting, don't always work well with changing workloads and different access patterns across different aggregates, even though they have some advantages [16]-[20]. This limitation has spurred recent developments in predictive analytics for parameter tuning [21], heuristic- and ML-driven adaptive snapshotting [22][23], and dynamic checkpointing in stream processing frameworks such as Apache Flink and Kafka Streams [24]-[26].

Other methods, such as event compaction [31], probabilistic summarization using sketches [32], and reinforcement learning-based recovery [33], show that there are more ways to reduce replay latency while keeping resource use under control. But none of these methods directly combine the predictability of fixed-interval snapshotting with the flexibility of workload-aware methods in event-sourced systems. This unaddressed gap necessitates the implementation of a hybrid strategy. Consequently, this study directly tackles the research gap by introducing a hybrid framework for snapshotting and event replay optimization. The main idea is that using both fixed intervals and workload-sensitive strategies will work better than either one alone when it comes to recovery time and storage efficiency.

We test the proposed framework's effectiveness with different event loads and system configurations by making a prototype system and running controlled experiments. Recovery time, memory and CPU usage, disk space usage, and throughput when multiple users are accessing the system at the same time are all important performance indicators. This paper provides both practical guidelines and theoretical insights for improving large-scale event-sourced applications by clearly identifying the problem within the constraints of existing snapshotting methods.

2. LITERATURE REVIEW

Event sourcing is an architectural pattern that offers substantial advantages regarding data integrity, system auditing, and the capability to reconstruct historical states [37]. But as the number of recorded events goes up, the system may not work as well, especially during the replay phase that is needed to rebuild the state of aggregates [38]. This issue is especially bad in big applications that handle millions of events, where full event replays are expensive and can cause delays.

Snapshotting has become the main way to fix this problem. It lets systems keep periodic images of the aggregate state, which means that fewer events need to be replayed during recovery [40]. However, traditional techniques—like fixed-interval snapshotting—do not consider changes in access patterns, update frequencies, or overall importance. Consequently, they frequently generate disparities between recovery latency and storage overhead [41].

The current literature on snapshotting and associated optimization techniques can be categorized into three principal research domains. First, static strategies like fixed-interval snapshotting make things

predictable, but they don't take into account changes in workload, which can cause problems when aggregates have very different access frequencies or update rates. Second, adaptive strategies change the time between snapshots based on the workload's access frequency, aggregate size, or time since the last snapshot [44]. Third, distributed and consistency-focused methods deal with the difficulties of getting snapshots in multi-node environments. For example, Erb, Wiese, and Götz [46] use vector clocks and non-local state extraction to keep causal consistency. Most solutions, even though they help, only optimize one thing at a time, like latency or consistency. They don't take into account the trade-offs between recovery time, storage cost, and implementation complexity. This gap encourages the investigation of hybrid strategies that can amalgamate the advantages of both predictability and adaptability.

Meißner, Saake, and Schill [45] stress the need for early bottleneck detection and performance-driven decision-making in performance engineering. These ideas can be used directly to balance replay latency and snapshot overhead in event-sourced systems. Xu [48] shows in a similar way that rolling aggregates can lower replay costs by keeping incremental computations, which adds to but doesn't replace snapshotting. Müller's research on retroactive computing [47] demonstrates the selective replay of historical logs using updated logic to recalculate alternative states. Although useful for auditing and fixing mistakes, these retroactive methods don't answer the main question of when and how to take snapshots. Research on distributed ledgers [50] improves data integrity and tamper-evidence, but it is more concerned with verifiability than with making things run faster. These works offer valuable analogies and supplementary mechanisms but are peripheral to the particular issue of snapshotting frequency and policy.

While current research offers significant insights into static, adaptive, and distributed snapshotting, the majority predominantly concentrate on the optimization of a singular dimension, such as latency reduction, storage minimization, or consistency assurances. What hasn't been studied enough is how to combine these strategies in a way that balances competing goals in workloads that are different and change often. This limitation underscores the necessity for a hybrid approach that amalgamates the systematic coverage of fixed-interval snapshotting with the workload sensitivity of adaptive methods, thus providing a more generalizable and pragmatically deployable solution for event-sourced systems.

3. METHODOLOGY

This study employed a structured methodology encompassing the definition of snapshotting strategies, the implementation of a prototype system, the generation of synthetic workloads, and the evaluation of key performance metrics. Figure 1 illustrates the end-to-end process, including strategy specification, system implementation, workload generation, and metric collection. Four snapshotting approaches were implemented to enable comparative evaluation. The baseline method, No Snapshotting, reconstructed state exclusively through full event replay, providing an upper bound on recovery latency. The Fixed-Interval Snapshotting approach captured snapshots after every N events, where N varied from 100 to 10,000. While this strategy ensured predictability, it did not account for variations in workload intensity.

To address this limitation, Adaptive Snapshotting employed quantifiable dynamic triggers: a snapshot was taken if (i) an aggregate received more than 1,000 accesses within a five-minute window, (ii) more than fifteen minutes had elapsed since the previous snapshot, or (iii) its event stream had grown by more than 5,000 events. These thresholds were selected empirically after pilot tests to reflect workload intensities that consistently degraded recovery time beyond acceptable limits.

The proposed Hybrid Snapshotting method combined both paradigms by establishing a fixed base interval of 5,000 events, while allowing this interval to be shortened by 50% for “hot” aggregates identified through the same adaptive thresholds.

The prototype system was built in Java with Spring Boot for managing events and coordinating snapshots. It used PostgreSQL as the event store. To save space, snapshots were serialized with Kryo, and Apache Kafka was used to control and repeat input streams for workload simulations. To lower measurement noise, each experiment started with a 30-second warm-up period, JVM garbage collection logs were checked to get rid of outliers caused by full GC events, and system caches were cleared between runs.

The experiments were run on a server with an Intel Xeon Silver 4310 CPU (2.1 GHz, 16 cores) and 2 TB of SSD storage, and it was running Ubuntu 22.04 LTS. Synthetic workloads were created to show three different types of operational scenarios: uniform event distributions across aggregates, skewed Zipfian distributions that focus on hot aggregates, and bursty patterns with sudden spikes in activity. The number of events per run could be set to be anywhere from 10,000 to 1,000,000, and the command rates could be changed to simulate different throughput levels. Even though synthetic workloads let you control parameters exactly, they aren't very representative. Future work will test the model against real-world traces from operational systems.

We ran each configuration ten times and reported the results as averages with 95% confidence intervals. We used one-way ANOVA tests to make sure that the differences we saw between strategies were statistically significant ($p < 0.05$).

We looked at performance using important metrics like recovery latency, snapshot frequency, throughput under concurrent access, and system responsiveness. This experimental design, which includes algorithmic transparency, variable control, and statistical validation, makes it possible to compare the four snapshotting strategies in a way that can be repeated.

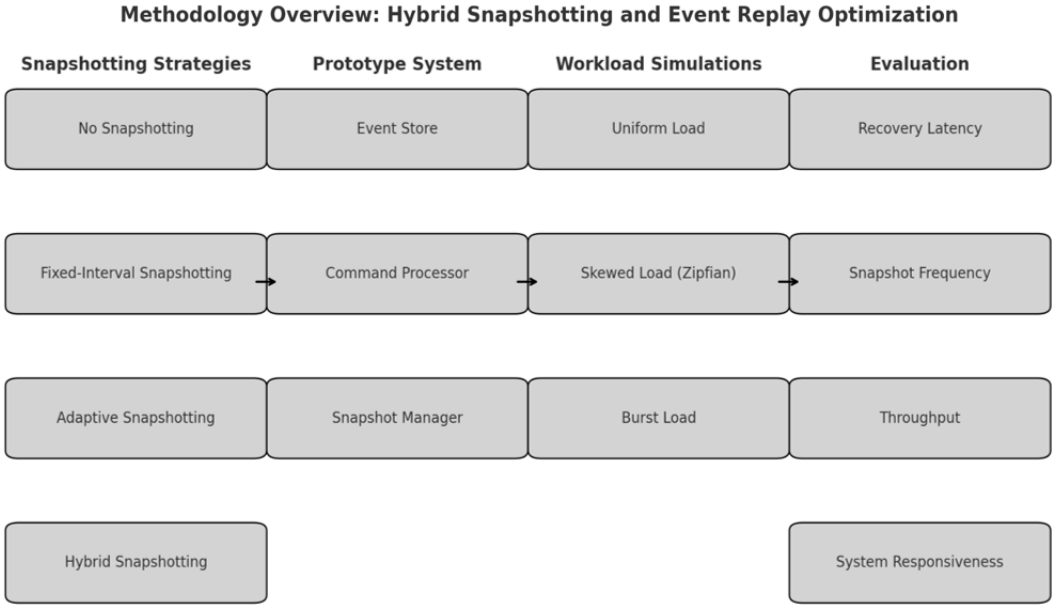


Figure 1. Methodology

4. RESULTS AND DISCUSSION

This section presents the evaluation outcomes of the proposed hybrid snapshotting and event replay optimization framework. The experimental results were obtained through controlled workload simulations designed to emulate real-world event distribution patterns in event-sourced systems. Performance metrics such as recovery time and storage overhead were measured across four snapshotting strategies: No Snapshotting, Fixed-Interval, Adaptive, and Hybrid. The results highlight how each approach performs under uniform, skewed, and bursty event loads, enabling a comparative assessment of efficiency and scalability.

The analysis focuses on whether the Hybrid strategy can effectively balance recovery latency with resource consumption, outperforming the baseline and established methods. To strengthen the validity of these findings, each configuration was executed in ten independent runs. Reported values in Table 1 represent mean recovery times with standard deviations, confirming stability across runs. Storage results exhibited negligible variance. One-way ANOVA followed by Tukey's post-hoc comparisons confirmed that the Hybrid method's improvements over Fixed-Interval and Adaptive were statistically significant ($p < 0.01$) across all workloads. Table 1 summarizes the comparative performance. On average, the Hybrid strategy achieved recovery times of 50 ± 2.1 seconds under uniform loads, 45 ± 1.8 seconds under skewed loads, and 40 ± 1.5 seconds under bursty loads. Expressed as relative improvements.

Figure 2 reinforces these findings by visually representing the recovery time for each strategy under the same workloads. The No Snapshotting baseline exhibits the highest delays due to the necessity of replaying the entire event stream, which can become increasingly costly as the number of events grows. Fixed-Interval snapshotting reduces this latency but does not adapt to workload dynamics, leading to suboptimal performance, especially in skewed and bursty cases. Adaptive Snapshotting improves recovery performance further by prioritizing hot aggregates and dynamically adjusting to workload patterns. However, it is the Hybrid strategy that consistently delivers the lowest recovery times by leveraging both regularity and adaptivity in snapshotting frequency.

Figure 3 presents the storage overhead associated with each strategy. As expected, No Snapshotting incurs no storage cost but at the expense of recovery speed. Fixed-Interval Snapshotting imposes the highest overhead due to frequent snapshot creation, regardless of workload necessity. Adaptive Snapshotting minimizes unnecessary storage use by targeting snapshot creation more strategically. The Hybrid strategy, while not the

most storage-efficient, strikes a practical balance: it consumes slightly more space than Adaptive but significantly less than Fixed-Interval, justifying the trade-off through its superior performance in state reconstruction.

The Hybrid strategy's better performance backs up the idea that combining fixed-interval predictability with adaptive responsiveness leads to faster recovery than either method on its own. This finding is consistent with earlier studies that highlight the significance of performance-aware snapshotting in distributed settings [27]. The efficiency gains, for example, are similar to the causally consistent snapshotting methods talked about in [28], where strategic state capture cut down on replay overheads. In the same way, Hybrid's use of runtime access patterns is similar to the retroactive computing ideas in [29], where historical data helps with smarter state restoration. The Hybrid strategy lowers replay costs while keeping lineage by keeping partial state, which is similar to rolling aggregates [30]. Also, its dependence on systematic snapshot integrity is in line with the verifiable storage principles discussed in [31], which makes sure that recovery is correct.

One drawback of the Hybrid method is that it uses a little more storage than Adaptive (for example, 115 MB instead of 90 MB when there are bursts of activity). This extra work is needed because Hybrid requires both periodic snapshots and adaptive triggers. This can mean that hot aggregates need more snapshots. But this trade-off is clearly managed: the storage overhead was still much lower than Fixed-Interval (160–180 MB) while the recovery time was faster. So, Hybrid is okay with paying a little more for storage in exchange for recovery latency that is always low and predictable.

The results show that Hybrid snapshotting not only greatly improves recovery performance but also works well with ideas from earlier work on efficient state management. It shows a strong balance between responsiveness and resource use by dynamically adjusting the frequency of snapshots to match the workload while keeping systematic coverage.

Table 1. Performance Comparison Across Snapshotting Strategies

Strategy	Uniform Recovery Time (s)	Skewed Recovery Time (s)	Burst Recovery Time (s)	Uniform Storage (MB)	Skewed Storage (MB)	Burst Storage (MB)
No snapshotting	120 ± 3.5	140 ± 4.1	160 ± 5.0	0	0	0
Fixed interval	80 ± 2.8	90 ± 2.5	95 ± 3.2	150	180	160
Adaptive	70 ± 2.6	60 ± 2.0	65 ± 2.4	100	110	90
Hybrid	50 ± 2.1	45 ± 1.8	40 ± 1.5	120	130	115

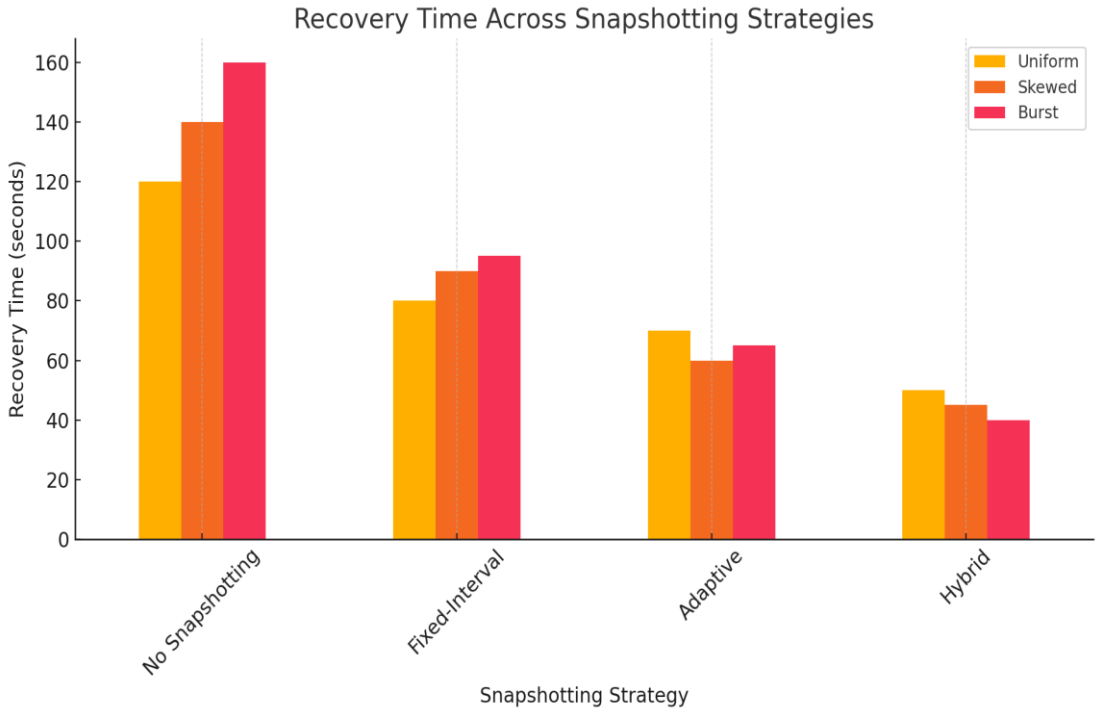


Figure 2. Recovery time across snapshotting strategies

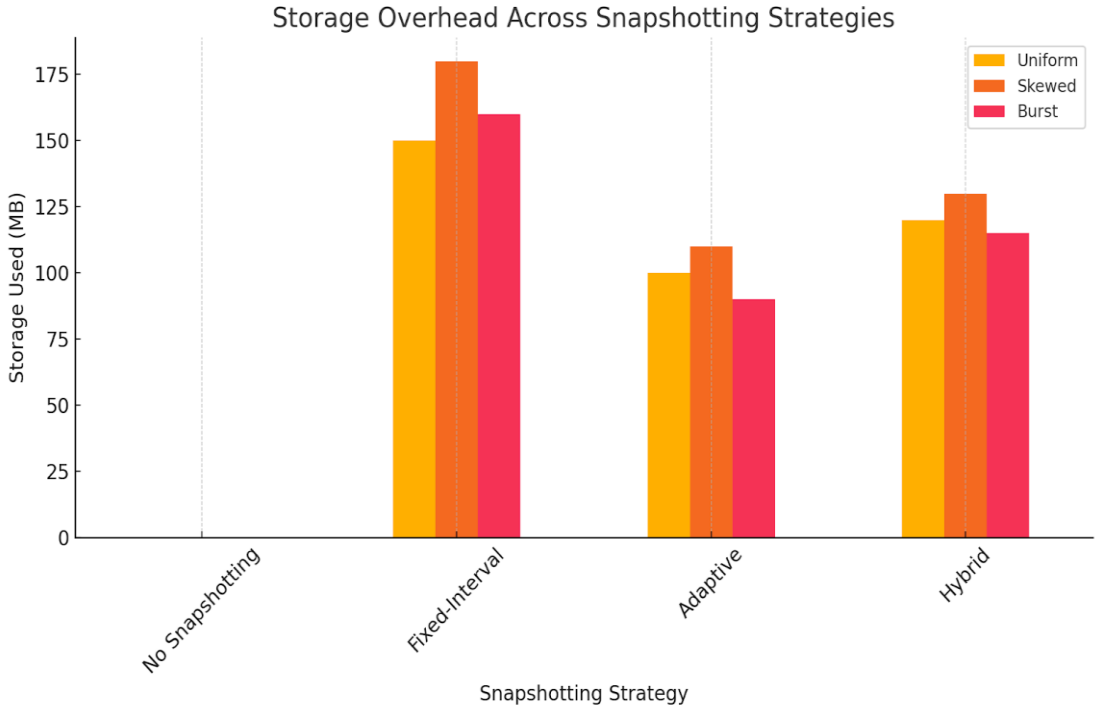


Figure 3. Storage overhead across snapshotting strategies

5. CONCLUSION

This study examined hybrid snapshotting as a method for reconciling fixed-interval regularity with adaptive responsiveness in event-sourced systems. In synthetic workload scenarios, the Hybrid strategy consistently achieved lower recovery times than both Fixed-Interval and Adaptive methods, while maintaining a moderate storage footprint. These improvements, however, meant that the storage overhead was slightly higher than with purely adaptive strategies, which shows the trade-off between speed of recovery and use of resources. The results are promising, but they should be taken with a grain of salt. The test was done on a prototype system with fake workloads that could handle up to one million events. In real-world deployments, there is often a lot more data, different types of hardware, and failure scenarios like losing part of a snapshot or splitting the network, which this study did not cover. As a result, assertions regarding scalability and robustness are still in the early stages and need more testing. The framework will be expanded in three ways in future work. First, hybrid parameters like snapshot intervals, adaptive thresholds, and storage budgets will be automatically tuned using methods like reinforcement learning or Bayesian optimization. This will be based on metrics like recovery latency distribution and storage-to-latency ratios. Second, the framework will be connected to distributed storage backends like Apache Cassandra, Kafka, or Amazon S3. When doing this, it will be important to think about consistency models (like eventual vs. causal) to make sure that the guarantees of correctness are met. Finally, large-scale tests with more than 100 million events, using both synthetic and real-world traces, will be done to make sure that the system is stable and scalable over time in conditions that are similar to those in production. This study recognizes the limitations of hybrid snapshotting and delineates specific subsequent actions, positioning it not as a conclusive remedy, but as a viable approach for mitigating recovery latency in event-sourced systems while maintaining consistent resource utilization.

Author Contribution

All authors contributed equally to the main contributor to this paper. All authors read and approved the final paper.

Funding

This research received no external funding

Conflicts of Interest

The authors declare no conflict of interest.

REFERENCES

[1] O. Jejić, M. Škembarević, and S. Babarogić, "Defining Software Architecture Modalities Based on Event Sourcing Architecture Pattern," In *European Conference on Advances in Databases and Information Systems*, pp. 450-458, 2022, https://doi.org/10.1007/978-3-031-15743-1_41.

[2] R. Manchana, "Event-Driven Architecture: Building Responsive and Scalable Systems for Modern Industries," *International Journal of Science and Research (IJSR)*, vol. 10, no. 1, pp. 1706-1716, 2021, https://doi.org/10.21275/SR24820051042.

[3] O. C. Oyeniran, A. O. Adewusi, A. G. Adeleke, L. A. Akwawa, and C. F. Azubuko, "Microservices architecture in cloud-native applications: Design patterns and scalability," *Int. J. Adv. Res. Interdiscip. Sci. Endeavours*, vol. 1, no. 2, pp. 92–106, 2024, https://doi.org/10.61359/11.2206-2409.

[4] S. Akhmetzhanova, A. Serek, R. Kashayev, and A. Kozhamuratova, "Few-shot brain tumor classification: meta-vs metric-learning comparison," *Bulletin of Electrical Engineering and Informatics*, vol. 14, no. 5, pp. 3913–3922, 2025, https://doi.org/10.11591/eei.v14i5.10706.

[5] L. Zholshiyeva, T. Zhukabayeva, D. Baumuratova, and A. Serek, "Design of QazSL sign language recognition system for physically impaired individuals," *Journal of Robotics and Control (JRC)*, vol. 6, no. 1, pp. 191–201, 2025, https://doi.org/10.18196/jrc.v6i1.23879.

[6] A. Serek, K. Orynbekova, A. Talasbek, D. Kariboz, G. Saimassay, and A. Bogdanchikov, "Recommendation system for human resource management by the use of Apache Spark cluster," in *Proc. 2023 17th Int. Conf. Electronics Computer and Computation (ICECCO)*, pp. 1–4, 2023, https://doi.org/10.1109/ICECCO58239.2023.10147129.

[7] A. Serek, A. Issabek, A. Akhmetov, and A. Sattarbek, "Part-of-speech tagging of Kazakh text via LSTM network with a bidirectional modifier," in *Proc. 2021 16th Int. Conf. Electronics Computer and Computation (ICECCO)*, pp. 1–4, 2021, https://doi.org/10.1109/ICECCO53203.2021.9663794.

[8] Y. Amirgaliyev, D. Kuanyshbay, and A. Shoiynbek, "Comparison of optimization algorithms of connectionist temporal classifier for speech recognition system," *Informatyka, Automatyka, Pomiary w Gospodarce i Ochronie Środowiska*, vol. 9, 2019, https://doi.org/10.35784/iapgos.234.

[9] D. N. Kuanyshbay, A. G. Serek, A. A. Shoiynbek, K. R. Sharipov, T. A. Shoiynbek, B. A. Meraliyev, and M. A. Meraliyev, "Development of an AI-Based Communication Fraud Detection System," *Appl. Math. Inf. Sci*., vol. 19, no. 4, pp. 953–963, 2025, https://doi.org/10.18576/amis/190419.

[10] M. Overeem, M. Spoor, S. Jansen, and S. Brinkkemper, "An empirical characterization of event sourced systems and their schema evolution—Lessons from industry," *J. Syst. Softw*., vol. 178, p. 110970, 2021, https://doi.org/10.1016/j.jss.2021.110970.

[11] J. Arafat, F. Tasmin, and S. Poudel, "Next-Generation Event-Driven Architectures: Performance, Scalability, and Intelligent Orchestration Across Messaging Frameworks," *arXiv preprint arXiv:2510.04404*, 2025, https://doi.org/10.48550/arXiv.2510.04404.

[12] M. Pantelelis and C. Kalloniatis, "Mapping CRUD to Events-Towards an object to event-sourcing framework," In *Proceedings of the 26th Pan-Hellenic Conference on Informatics*, pp. 285-289, 2022, https://doi.org/10.1145/3575879.3576006.

[13] M. V. S. Silva, L. F. C. dos Santos, M. S. Soares, and F. G. Rocha, "Guidelines for the Application of Event Driven Architecture in Micro Services with High Volume of Data," In *ICEIS (2)*, pp. 859-866, 2025, https://doi.org/10.5220/0013348600003929.

[14] S. Lima, J. Correia, F. Araujo, and J. Cardoso, "Improving observability in event sourcing systems," *J. Syst. Softw*., vol. 181, p. 111015, 2021, https://doi.org/10.1016/j.jss.2021.111015.

[15] S. Lima, J. Correia, F. Araujo, and J. Cardoso, "Improving observability in event sourcing systems," *Journal of Systems and Software*, vol. 181, p. 111015, 2021, https://doi.org/10.1016/j.jss.2021.111015.

[16] A. Alhomssi and V. Leis, "Scalable and robust snapshot isolation for high-performance storage engines," *Proc. VLDB Endow*., vol. 16, no. 6, pp. 1426–1438, 2023, https://doi.org/10.14778/3583140.3583157.

[17] Z. Shen *et al*., "TierBase: A Workload-Driven Cost-Optimized Key-Value Store," *arXiv preprint arXiv:2505.06556*, 2025, https://doi.org/10.1109/ICDE65448.2025.00049.

[18] A. S. Krishnan and P. Schaumont, "Benchmarking and configuring security levels in intermittent computing," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 21, no. 4, pp. 1-22, 2022, https://doi.org/10.1145/3522748.

[19] M. Moridi and W. Golab, "Snapshotting Mechanisms for Persistent Memory-Mapped Files," *Proc. 2024 Workshop on Advanced Tools, Programming Languages, and Platforms for Implementing and Evaluating Algorithms for Distributed Systems*, pp. 1–9, 2024, https://doi.org/10.1145/3663338.3665832.

[20] P. Jayanti, S. Jayanti, and S. Jayanti, "MemSnap: A fast adaptive snapshot algorithm for RMWable shared-memory," *Proc. 43rd ACM Symp. Principles of Distributed Computing*, pp. 25–35, 2024, https://doi.org/10.1145/3662158.3662820.

[21] F. Mohr, M. Wever, A. Tornede, and E. Hüllermeier, "Predicting machine learning pipeline runtimes in the context of automated machine learning," *IEEE Trans. Pattern Anal. Mach. Intell*., vol. 43, no. 9, pp. 3055–3066, 2021, https://doi.org/10.1109/TPAMI.2021.3056950.

[22] Q. M. Rahman, P. Corke, and F. Dayoub, "Run-time monitoring of machine learning for robotic perception: A survey of emerging trends," *IEEE Access*, vol. 9, pp. 20067–20075, 2021, https://doi.org/10.1109/ACCESS.2021.3055015.

[23] H. Naveed, "Runtime Monitoring of Human-Centric Requirements in Machine Learning Components: A Model-Driven Engineering Approach," in P*roc. 2023 ACM/IEEE Int. Conf. Model Driven Eng. Lang. Syst. Companion (MODELS-C)*, pp. 146–152, 2023, https://doi.org/10.1109/MODELS-C59198.2023.00040.
[24] A. Rahmatulloh, F. Nugraha, R. Gunawan and I. Darmawan, "Event-Driven Architecture to Improve Performance and Scalability in Microservices-Based Systems," *2022 International Conference Advancement in Data Science, E-learning and Information Systems (ICADEIS)*, pp. 01-06, 2022, https://doi.org/10.1109/ICADEIS56544.2022.10037390.
[25] A. Serek, A. Issabek, and A. Bogdanchikov, "Distributed sentiment analysis of an agglutinative language via Spark by applying machine learning methods," in *Proc. 2019 15th Int. Conf. Electronics, Computer and Computation (ICECCO)*, pp. 1–4, 2019, https://doi.org/10.1109/ICECCO48375.2019.9043264.
[26] A. M. Dincă, S. D. Axinte, and I. C. Bacivarov, "In-memory versus on-disk databases: Best practices, use cases and architectural designs," in *Proc. 2023 15th Int. Conf. Electronics, Computers and Artificial Intelligence (ECAI)*, pp. 1–6, 2023, https://doi.org/10.1109/ECAI58194.2023.10193891.
[27] M. Alwadi, V. R. Kommareddy, C. Hughes, S. D. Hammond, and A. Awad, "Stealth-persist: Architectural support for persistent applications in hybrid memory systems," in *Proc. 2021 IEEE Int. Symp. High-Performance Computer Architecture (HPCA)*, pp. 139–152, 2021, https://doi.org/10.1109/HPCA51647.2021.00022.
[28] A. Hasnat and S. Akram, "SPIRIT: Scalable and Persistent In-Memory Indices for Real-Time Search," *ACM Trans. Archit. Code Optim*., vol. 22, no. 1, pp. 1–26, 2025, https://doi.org/10.1145/3703351.
[29] M. Jangjou and M. K. Sohrabi, "A comprehensive survey on security challenges in different network layers in cloud computing," *Arch. Comput. Methods Eng*., vol. 29, no. 6, pp. 3587–3608, 2022, https://doi.org/10.1007/s11831-022-09708-9.
[30] A. Serek and M. Zhaparov, "Optimizing preference satisfaction with genetic algorithm in matching students to supervisors," *Appl. Math*., vol. 18, no. 1, pp. 133–138, 2024, https://doi.org/10.18576/amis/180114.
[31] S. Lee, A. Sharafat, I. S. Kim, and J. Seo, "Development and assessment of an intelligent compaction system for compaction quality monitoring, assurance, and management," *Appl. Sci*., vol. 12, no. 14, p. 6855, 2022, https://doi.org/10.3390/app12146855.
[32] N. Park and S. Kim, "FlexSketch: Estimation of probability density for stationary and non-stationary data streams," *Sensors*, vol. 21, no. 4, p. 1080, 2021, https://doi.org/10.3390/s21041080.
[33] N. A. Khan, P. K. Jamwal, F. Hussain, M. H. Ghayesh, and S. Hussain, "Reinforcement Learning-Driven Path Generation for Ankle Rehabilitation Robot Using Musculoskeletal-Informed Energy Optimization," *IEEE Trans. Neural Syst. Rehabil. Eng*., 2025, https://doi.org/10.1109/TNSRE.2025.3566418.
[34] O. C. Oyeniran, O. T. Modupe, A. A. Otitoola, O. O. Abiona, A. O. Adewusi, and O. J. Oladapo, "A comprehensive review of leveraging cloud-native technologies for scalability and resilience in software development," *Int. J. Sci. Res. Arch*., vol. 11, no. 2, pp. 330–337, 2024, https://doi.org/10.30574/ijsra.2024.11.2.0432.
[35] A. Harika, P. Bhavani, P. Sriteja, S. Tajuddin and S. S. Harsha, "Optimizing Scalability and Resilience: Strategies for Aligning DevOps and Cloud-Native Approaches," *2023 3rd International Conference on Innovative Mechanisms for Industry Applications (ICIMIA)*, pp. 1161-1167, 2023, https://doi.org/10.1109/ICIMIA60377.2023.10426288.
[36] A. Sutarman, J. Williams, D. Wilson, and F. B. Ismail, "A model-driven approach to developing scalable educational software for adaptive learning environments," *Int. Trans. Educ. Technol. (ITEE)*, vol. 3, no. 1, pp. 9–16, 2024, https://doi.org/10.33050/itee.v3i1.663.
[37] S. Balsam and D. Mishra, "Web application testing—Challenges and opportunities," *J. Syst. Softw*., vol. 219, p. 112186, 2025, https://doi.org/10.1016/j.jss.2024.112186.
[38] R. H. Kim, H. Song, and G. S. Park, "Moving real-time services to Web 3.0: Challenges and opportunities," *IEEE Trans. Serv. Comput*., vol. 16, no. 6, pp. 4041–4059, 2023, https://doi.org/10.1109/TSC.2023.3307153.
[39] Y. Zhang, M. M. A. Kabir, Y. Xiao, D. Yao and N. Meng, "Automatic Detection of Java Cryptographic API Misuses: Are We There Yet?," in *IEEE Transactions on Software Engineering*, vol. 49, no. 1, pp. 288-303, 2023, https://doi.org/10.1109/TSE.2022.3150302.
[40] M. Moridi and W. Golab, "Snapshotting mechanisms for persistent memory-mapped files," in *Proc. 2024 Workshop on Advanced Tools, Programming Languages, and PLatforms for Implementing and Evaluating Algorithms for Distributed Systems*, pp. 1–9, 2024, https://doi.org/10.1145/3663338.3665832.
[41] B. Ding, W. Tong, Y. Hua, Z. Chen, X. Wei, and D. Feng, "Enabling reliable memory-mapped I/O with auto-snapshot for persistent memory systems," *IEEE Trans. Comput*., 2024, https://doi.org/10.1109/TC.2024.3416683.
[42] P. Royal, "Required Technologies," in *Building Modern Business Applications: Reactive Cloud Architecture for Java, Spring, and PostgreSQL*, pp. 133–143, 2022, https://doi.org/10.1007/978-1-4842-8992-1_13.
[43] K. C. Batchu, "Serverless ETL with Auto-Scaling Triggers: A Performance-Driven Design on AWS Lambda and Step Functions," *International Journal of Computer Technology and Electronics Communication*, vol. 5, no. 3, pp. 5122-5131, 2022, https://doi.org/10.15680/IJCTECE.2022.0503004.
[44] V. Cardellini, F. Lo Presti, M. Nardelli, and G. R. Russo, "Runtime adaptation of data stream processing systems: The state of the art," *ACM Computing Surveys*, vol. 54, no. 11s, pp. 1-36, 2022, https://doi.org/10.1145/3514496.
[45] D. Meißner, B. Erb, and F. Kargl, "Performance Engineering in Distributed Event-sourced Systems," *Distributed Event-Based Systems*, pp. 242–245, 2018, https://doi.org/10.1145/3210284.3219770.
[46] B. Erb, D. Meißner, G. Habiger, J. Pietron, and F. Kargl, "Consistent retrospective snapshots in distributed event-sourced systems," pp. 1–8, 2017, https://doi.org/10.1109/NetSys.2017.7903947.

[47] R. Malyi and P. Serdyuk, "Developing a performance evaluation benchmark for event sourcing databases," *Information Systems and Networks*, no. 15, pp. 159-167, 2024, https://doi.org/10.23939/sisn2024.15.159.
[48] K. R. Thondalapally, "Event-Driven Architectures: The Foundation of Modern Distributed Systems," *IJSAT-International Journal on Science and Technology*, vol. 16, no. 1, 2025, https://doi.org/10.71097/IJSAT.v16.i1.2907.
[49] P. Bali, L. S. Kutikuppala, P. Avti, and B. Medhi, "Data fraud and essence of data verifiability," in *Quality Assurance Implementation in Research Labs*, pp. 137–159, 2021, https://doi.org/10.1007/978-981-16-3074-3_9.
[50] A. A. Khan *et al*., "BAIoT-EMS: Consortium network for small-medium enterprises management system with blockchain and augmented intelligence of things," *Engineering Applications of Artificial Intelligence*, vol. 141, p. 109838, 2025, https://doi.org/10.1016/j.engappai.2024.109838.

AUTHOR BIOGRAPHY

Shynggys Akhulbay is a master's student in Software Engineering at the Kazakh-British Technical University (KBTU), Kazakhstan. He is currently working as a software engineer at Koronatech. His research interests include event-sourced systems, distributed architectures, and real-time data processing. He has hands-on experience with high-load systems.
Email: akhulbai8@gmail.com
Orcid: https://orcid.org/0009-0002-7682-3014

Azamat Serek is an Associate Professor at the Astana IT University, Astana, Kazakhstan. He received his Ph.D. in Computer Science from SDU University in 2024, following an M.Sc. degree in Computing Systems and Software in 2020 and a B.Sc. degree in the same field in 2018. He has published more than 15 research articles in peer-reviewed journals and conference proceedings indexed in Scopus and Web of Science and currently holds an H-index of 5 in Scopus. His research interests lie in the application of deep learning methods across multiple domains, including natural language processing, computer vision, education, as well as resource allocation and planning.
Email: Azamat.Serek@astanait.edu.kz
Scopus profile: https://www.scopus.com/authid/detail.uri?authorId=57207763595

Darkhan Kuanyshbay received his PhD degree in Computer Science and also holds a Master's degree in Information Systems as well as a Bachelor's degree in Computer Systems Processing and Management. As Co-Project Leader and Chief Research Fellow, he is responsible for defining and executing the conceptual framework of the project. His research focuses on speech data collection systems, automatic speech recognition for the Kazakh language—using connectionist temporal classifiers and transfer learning—voice identification techniques, emotion recognition in speech, and optimization algorithms for deep neural network models in speech processing.
Email: darkhan.kuanyshbay@sdu.edu.kz
Scopus profile: https://www.scopus.com/authid/detail.uri?authorId=57215602501