UDC 004.62:004.75
IRSTI 50.41.21; 28.21.27

https://doi.org/10.55452/1998-6688-2026-23-2-159-170

[1]Onayeva A.E.,
MSc., ORCID ID: 0009-0004-4379-1065
e-mail: al_onayeva@kbtu.kz

[1]Kazakh-British Technical University, Almaty, Kazakhstan

A THREE-TIER INFORMATION LIFECYCLE MANAGEMENT MECHANISM FOR HOT-TIER MEMORY CONTROL IN CONTAINERIZED KAFKA–FLINK STREAMING PIPELINES

Abstract

Real-time streaming systems face persistent memory pressure as continuous data ingestion drives unbounded growth in hot-tier storage. Existing Information Lifecycle Management (ILM) frameworks have been applied primarily to enterprise archival contexts and have not been evaluated within containerized streaming pipelines that employ multi-tier in-memory architectures. This paper presents a lightweight, policy-driven ILM mechanism integrated into a Kafka–Apache Flink pipeline with a three-tier storage model comprising MongoDB (hot-tier), TimescaleDB (warm-tier), and Parquet files (cold-tier). An asynchronous sweeper thread migrates records between tiers according to configurable time thresholds and , preventing hot-tier saturation without disrupting stream processing. Five experiments were conducted to evaluate memory efficiency, tier retrieval latency, threshold sensitivity, scalability, and extended lifecycle behavior. The results demonstrate that ILM reduces MongoDB peak memory usage by 81% (from 106.96 ± 1.63 MB to 20.28 ± 1.81 MB, $p < 0.001$) while Flink throughput and processing latency remain unaffected. Memory bounds hold stably across ingestion rates from 200 to 1,000 messages per second. An extended 90-minute run validates correct three-tier lifecycle operation, with MongoDB remaining bounded, TimescaleDB absorbing 2.28 million warm records, and Parquet accumulating cold archives. These findings confirm that effective, low-overhead ILM can be achieved in containerized real-time pipelines using only native database capabilities and file system operations.

Keywords: Information Lifecycle Management (ILM), Apache Kafka, Apache Flink, MongoDB, TimescaleDB, real-time streaming, memory optimization, tiered storage, containerized pipelines, data archiving.

Received November 11, 2025; revised April 25, May 1, 2026; accepted May 4, 2026.

Introduction

In the current era of digital transformation, data has become the core of enterprise development and strategic competition [1]. The explosive growth of data traffic, driven by IoT sensors, social networks, and mobile applications, is estimated to double every two years, with global volumes projected to reach 221.2 ZB by 2026 [1–3]. As these numbers grow, traditional data management approaches have begun to show significant performance degradation. This challenge has triggered the development of Information Lifecycle Management (ILM), a comprehensive methodological strategy that reduces inefficiencies and optimizes data management by strategically aligning information resources with crucial business objectives and operational requirements.

As data expands in volume, velocity, and variety (3V's), and as extended by to include veracity, variability, and value, organizations face escalating costs and operational bottlenecks [4]. ILM has become a critical methodology to address these challenges by systematically managing data throughout its lifecycle. It ensures that high-value data remains cost-effectively accessible while redundant or low-value information is eliminated early, thereby improving both efficiency and governance.

ILM research has emphasized automation and policy-driven data movement. Govil et al. highlight that automated lifecycle policies reduce manual intervention and improve productivity by aligning data storage with business needs [5]. Similarly, Salutina et al. and Ahmad and Rai discuss how digital transformation and big data frameworks demand advanced ILM strategies to sustain performance and scalability in dynamic environments [6, 7]. These studies collectively underscore ILM's importance in structuring, classifying, and optimizing massive data systems.

Further investigations demonstrate ILM's versatility across different domains. Wang applied algorithmic policies such as capacity migration to improve storage utilization, while Shen developed hierarchical storage layers to balance access speed and cost [8, 9]. In specialized contexts, some studies extend ILM principles to mission-critical and secure data lake environments, integrating techniques like anomaly detection and machine learning to enhance lifecycle governance and data quality [10–12].

By utilizing Hierarchical Storage Management (HSM), organizations grade data based on utility and migrate it across tiers: the hot tier holds frequently accessed, time-sensitive data in high-performance storage; the warm tier handles data with decreasing access frequency on mid-tier devices; and the cold tier provides low-cost, high-capacity archival storage for long-term retention and regulatory compliance.

Despite this substantial body of work, a critical gap remains: ILM has not been applied to real-time streaming pipelines in which hot-tier in-memory stores face rapid saturation under continuous message ingestion. In such systems, data arrives at high frequency, and the absence of automated lifecycle policies causes in-memory databases to grow unboundedly, eventually exhausting available resources. Existing ILM frameworks are designed for batch-oriented or enterprise environments and do not address the asynchronous, low-latency requirements of streaming architectures. This gap motivates our study, which introduces a lightweight, container-native ILM mechanism for a Kafka–Apache Flink streaming pipeline with a three-tier storage hierarchy.

Materials and methods

System Architecture

The proposed system is a fully containerized streaming pipeline that combines data ingestion, real-time processing, tiered storage, and lifecycle management within a single Docker Compose deployment. The architecture comprises four functional layers: ingestion, processing, storage, and analytics, as shown in Figure 1.

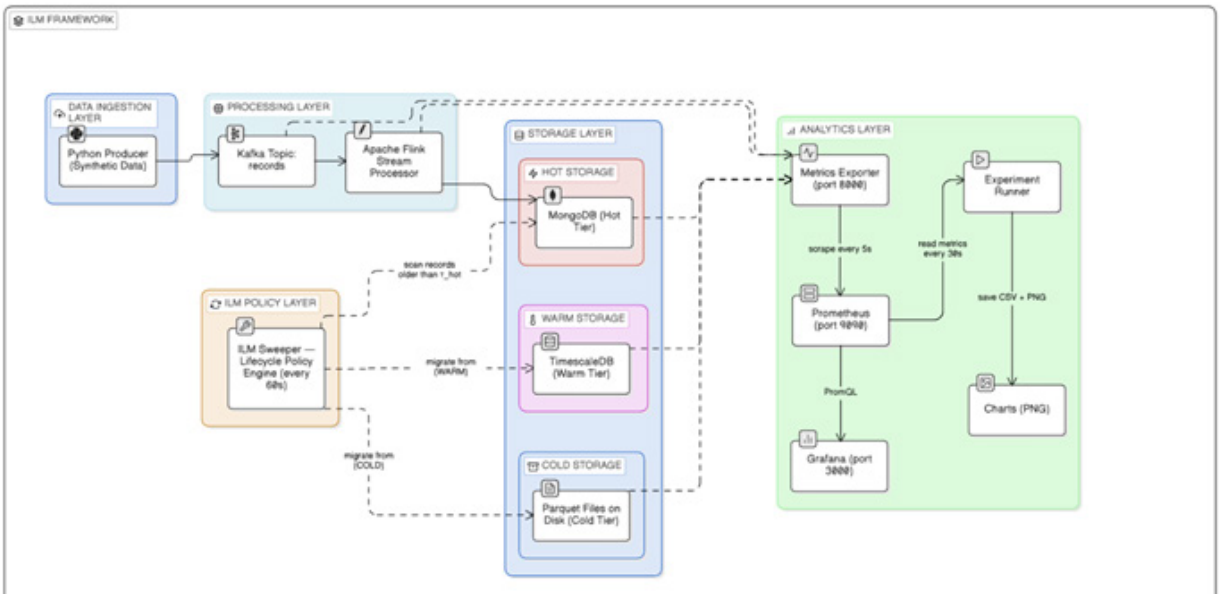

Figure 1 – High-level architecture of the proposed ILM-driven streaming pipeline

The ingestion layer consists of a Python-based Kafka producer that streams synthetically generated records at a configurable message rate (default 300 msg/s). The processing layer employs Apache Flink, which consumes messages from Kafka and routes each record to the appropriate storage tier based on its age. The storage layer implements a three-tier hierarchy: MongoDB (hot tier) holds recently ingested records requiring low-latency access, TimescaleDB (warm tier) stores records that have aged beyond , and Parquet files on disk (cold tier) serve as long-term archives for records exceeding τ_cold. The analytics layer monitors memory usage, throughput, and latency via Prometheus and a background metrics exporter.

Apache Kafka

Apache Kafka (Confluent Platform 7.2.1) serves as the message ingestion buffer. Built on an append-only distributed log architecture, Kafka provides reliable message persistence, ordered delivery within partitions, and the ability to replay messages from any offset [13]. In the proposed system, a single Kafka topic receives streaming metadata from the Python producer. Kafka decouples the producer from the downstream Flink consumer, absorbing bursts in ingestion rate without backpressure propagation.

Apache Flink

Apache Flink (version 1.17) serves as the stream processing engine. Flink implements a unified dataflow model that treats both streaming and batch workloads as pipelined, fault-tolerant dataflows [14]. Unlike micro-batch systems, Flink processes records as a true continuous stream, enabling consistently low processing latency. Flink's stateful operators, native Kafka source connector, and exactly-once delivery semantics make it suitable for the ILM classification task, which requires correct record routing even in the presence of processing failures [15]. The Flink job reads records from Kafka, attaches an insertion timestamp to each message, applies the ILM classification tag, and writes the record to MongoDB.

MongoDB

MongoDB (version 6.0) functions as the hot-tier store. As a document-oriented NoSQL database, MongoDB stores records as BSON documents in memory-mapped collections, enabling sub-millisecond read and write operations for recently ingested data. Each record is keyed by a composite identifier derived from the record_id field and ingestion timestamp, enabling efficient age-based queries during sweeper cycles. Memory consumption is tracked using the serverStatus.mem. resident counter sampled at one-second intervals.

TimescaleDB

TimescaleDB (version 2.11 on PostgreSQL 15) serves as the warm-tier store. Its time-series hypertable architecture automatically partitions data into fixed-interval chunks, ensuring that recent data remains in memory while older chunks are accessed from disk [16]. This design provides efficient insertion and range-query performance on time-ordered records. The hypertable is partitioned on the record insertion timestamp at one-hour intervals, matching the threshold used in the experiments.

Apache Parquet

Parquet files on a local Docker volume function as the cold tier. Apache Parquet is a columnar storage format that provides high compression ratios through column-level encoding and predicate pushdown during batch scans [17]. Records are serialized as gzip-compressed Parquet files organized by a timestamp-based directory hierarchy (year/month/day/hour). Parquet imposes no ongoing memory cost, requires no running database process, and is natively supported by major batch processing frameworks.

ILM Mechanism

The ILM mechanism classifies each record based on its age relative to two configurable thresholds: the hot threshold and the cold threshold . A record written to MongoDB at insertion time is eligible for warm-tier migration at time t when the following condition is satisfied:

$$t - t_k \geq \tau_{\text{hot}} \quad (1)$$

Similarly, a record stored in TimescaleDB at time becomes eligible for cold archiving when:

$$t - t_j \geq \tau_{cold} \tag{2}$$

An asynchronous sweeper thread executes Algorithm 1 at a fixed interval of 60 seconds throughout each experimental run. The sweeper operates entirely independently of the Flink processing job, introducing no latency on the critical stream processing path.

Algorithm 1: ILM Sweeper Thread

Algorithm 1 ILM Sweeper Thread

Require: τ_{hot}, τ_{cold}, $sweep\_interval = 60$ s
Ensure: Migrated records across hot → warm → cold tiers
repeat
  $t_{now} \leftarrow$ current Unix timestamp
  {Hot → Warm migration}
  $E_{hot} \leftarrow$ **query** MongoDB WHERE $(t_{now} - t_k) \geq \tau_{hot}$
  if $E_{hot} \neq \emptyset$ **then**
    bulk insert E_{hot} INTO TimescaleDB
    delete E_{hot} FROM MongoDB
  end if
  {Warm → Cold migration}
  $E_{cold} \leftarrow$ **query** TimescaleDB WHERE $(t_{now} - t_j) \geq \tau_{cold}$
  if $E_{cold} \neq \emptyset$ **then**
    serialize E_{cold} → Parquet file (gzip, timestamp-partitioned)
    delete E_{cold} FROM TimescaleDB
  end if
  sleep($sweep\_interval$)
until pipeline stopped =0

Figure 2 illustrates the characteristic sawtooth pattern produced by the sweeper cycle: memory rises as new records accumulate in MongoDB between sweeps, then drops sharply when the sweeper migrates eligible records to TimescaleDB.

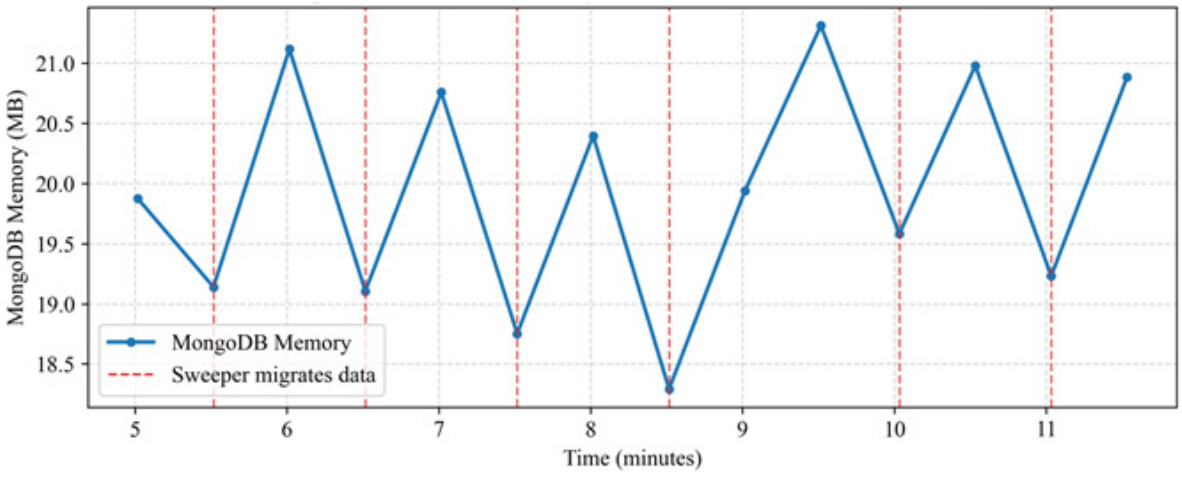

Figure 2 – Sweeper cycle effect on MongoDB memory

Each Kafka message is a JSON payload containing four fields: a unique record identifier, a Unix epoch timestamp added at ingestion time, a random floating-point value, and a categorical label. The Unix epoch timestamp serves as the lifecycle clock for all ILM age calculations.

Table 1 – Fields of a Kafka message

Field	Example Value
record_id	3f7a2c1d-9e4b-4a1f-b8c2-0d5e6f7a8b9c
timestamp	1747498811.13
value	0.7342891
category	analytics

Experimental Setup

All experiments were conducted on a Lenovo Yoga Slim 7 Pro 14ACH5 equipped with an AMD Ryzen 7 5000-series processor (8 cores, up to 4.4 GHz), 16 GB of RAM, and a 1 TB NVMe solid-state drive. The host operating system is Windows 11 with Docker Desktop (version 4.40.0) and WSL2 integration. All pipeline components – Kafka, Flink, MongoDB, TimescaleDB, and the ILM sweeper – run inside Docker containers on an Ubuntu-based backend. Python 3.11 was used for the Kafka producer and ILM sweeper scripts.

Table 2 summarizes the five experimental configurations. Experiments 1 and 1_1 evaluate the core ILM memory benefit under standard and extended run conditions respectively. Experiment 2 quantifies cross-tier retrieval latency. Experiment 3 examines sensitivity to the τ_hot threshold, and Experiment 4 assesses scalability under varying ingestion rates.

Table 2 – Experimental settings summary

Parameter	Exp. 1	Exp. 2	Exp. 3	Exp. 4	Exp. 1_1
Duration	30 min	30 min	20 min	20 min	90 min
Repetitions	5	5	5 per τ	5 per rate	1
Rate (msg/s)	300	300	300	200/500/1000	300
τ_hot	300 s	300 s	60/180/300/600 s	300 s	300 s
τ_cold	3600 s	3600 s	3600 s	3600 s	2700 s
Sweeper	ON/OFF	ON	ON	ON	ON
Statistics	Paired t-test	mean ± std	mean ± std	mean ± std	Descriptive

Evaluation Metrics

Four metrics are used to evaluate ILM performance:

- MongoDB Peak Memory (MB) is the maximum resident memory consumed by MongoDB, measured via the serverStatus.mem.resident counter, quantifying the effectiveness of hot-tier memory offloading.
- Flink Throughput (rps) is the number of records processed per second by the Flink job, confirming that ILM introduces no overhead on the stream processing path.
- Flink Processing Latency (p95, ms) is the 95th percentile of per-record processing time, assessing tail latency behavior.
- Tier Retrieval Latency (ms) is the end-to-end query response time when fetching a single record from each storage tier, characterizing the speed-cost tradeoff of the three-tier architecture.

Results and discussion

Experiment 1: Baseline vs. ILM

Experiment 1 compares two configurations at 300 msg/s over 30 minutes: a baseline condition in which the ILM sweeper is disabled and MongoDB retains all records indefinitely, and an ILM-enabled condition in which the sweeper migrates records older than = 300 s to TimescaleDB. Five independent runs were conducted per condition and a paired t-test was applied to peak MongoDB memory measurements.

As can be seen in Table 3, MongoDB memory grew linearly, with no upper bound, from approximately 2.8 MB at t = 0 to 106.96 ± 1.63 MB at t = 1,800 s when ILM was disabled. During the initial warm-up, memory with ILM enabled reached about 20 MB, and for the next 25 minutes, it fluctuated between 18 and 22 MB. The memory trajectories for every run in both scenarios are displayed in Fig. 3. All three key measurements are shown in parallel with error bars in Figure 4.

Table 3 – Experiment 1: Baseline vs. ILM results

Metric	No ILM	With ILM	Effect
MongoDB Memory (MB)	106.96 ± 1.63	20.28 ± 1.81	81% reduction (5.3×)
Flink Throughput (rps)	467 ± 3.1	471 ± 7.4	No impact
Flink Latency p95 (ms)	1.0	1.0	No impact
p-value (t-test)	—	—	p = 0.000058

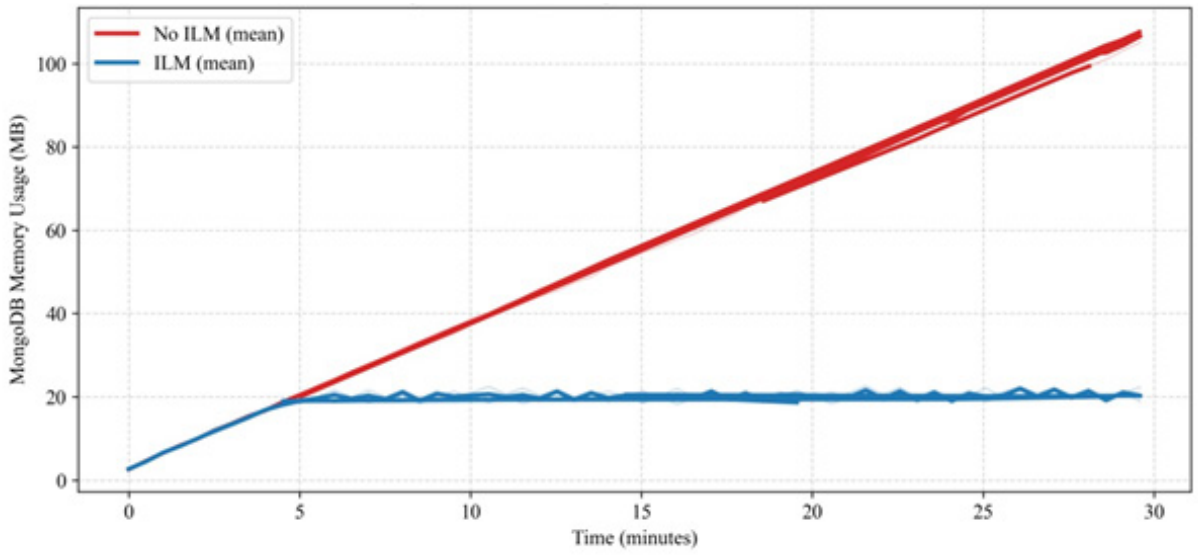

Figure 3 – MongoDB memory usage over 30 minutes

The 81% memory reduction (5.3× improvement) is statistically highly significant (p = 0.000058, far below α = 0.05). The low standard deviations (±1.63 and ±1.81 MB) confirm high reproducibility across five independent runs. Flink throughput remained virtually identical between conditions (467 vs. 471 rps) and p95 latency was unchanged at 1.0 ms, confirming that the ILM sweeper thread runs with negligible interference to the Flink stream processing job.

Experiment 2: Three-Tier Retrieval Latency

Experiment 2 measures the end-to-end query latency of retrieving a single record from each storage tier during a 30-minute run with all tiers active. A query agent issues one retrieval request per tier every five minutes.

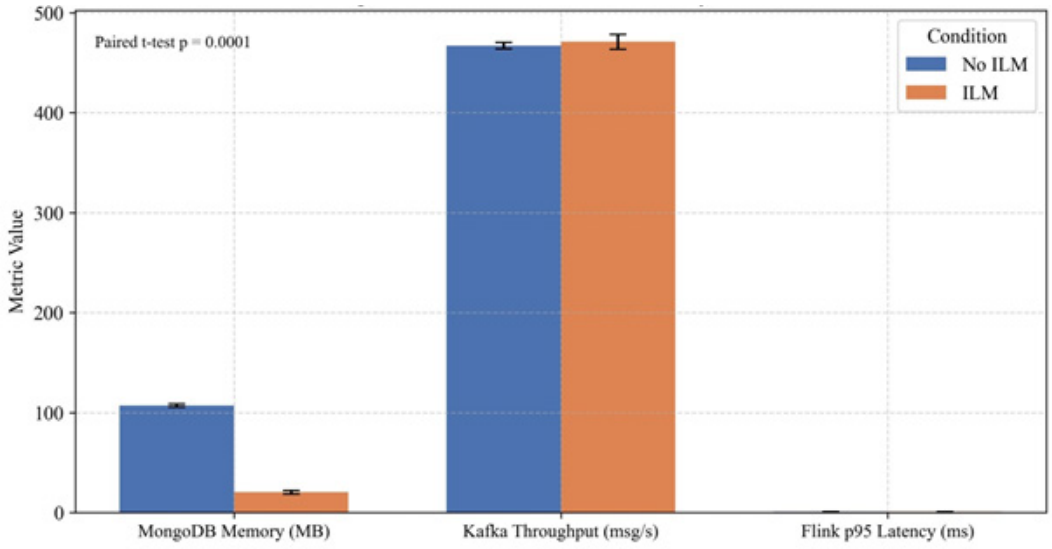

Figure 4 – Experiment 1 summary: key metrics with error bars

Results can be seen in Table 4 and Figure 5. MongoDB (hot tier) delivered queries with a mean latency of 28.9 ± 10.1 ms. TimescaleDB (warm tier) exhibited a mean latency of 677.5 ± 84.9 ms; this figure includes Docker exec overhead and should be interpreted as end-to-end access latency rather than raw database query time. The order-of-magnitude difference between hot and warm tiers is consistent with the access-cost tradeoff inherent in tiered storage. Cold-tier latency was not measured in this experiment as was not reached within the 30-minute window; cold-tier behavior is validated in Experiment 1_1.

Table 4 – Experiment 2: Tier retrieval latency

Tier	Mean Latency (ms)	Std (ms)
Hot (MongoDB)	28.9	±10.1
Warm (TimescaleDB)	677.5	±84.9

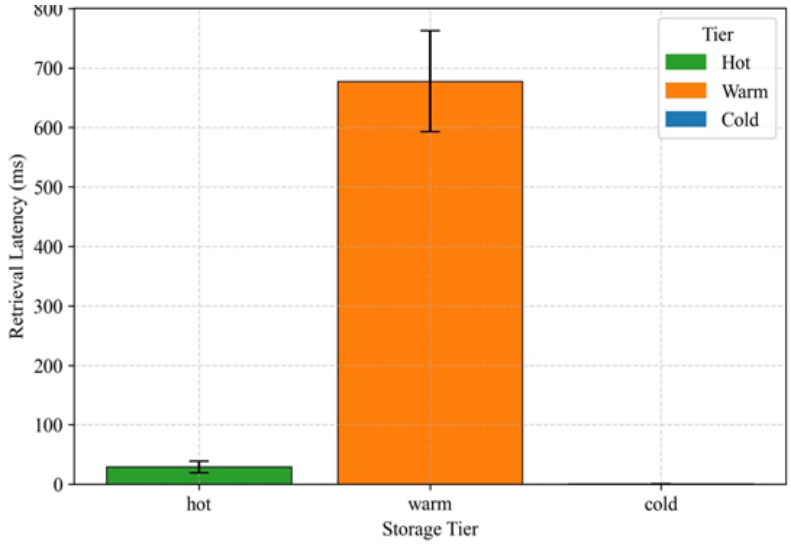

Figure 5 – Experiment 2: retrieval latency by storage tier

Experiment 3: Threshold Sensitivity

Experiment 3 investigates the effect of varying on MongoDB memory usage and hot-tier retrieval latency at 300 msg/s. Four threshold values – 60 s, 180 s, 300 s, and 600 s – were each tested over five independent 20-minute runs.

Table 5 and Figure 6 demonstrate that hot-tier latency (26.5–28.1 ms) and MongoDB memory (17.91–18.58 MB) remained almost constant across all threshold settings. There was no apparent statistically significant difference. This robustness results from the sweeper's ability to keep up with all arrival rates at 300 msg/s, independent of threshold selection, simplifying operational configuration.

Table 5 – Experiment 3: Threshold sensitivity results

τ_hot (s)	MongoDB Memory (MB)	Hot Latency (ms)
60	18.46 ± 5.39	27.1 ± 11.5
180	18.58 ± 5.49	28.1 ± 11.6
300	18.14 ± 5.28	27.7 ± 8.2
600	17.91 ± 5.29	26.5 ± 9.0

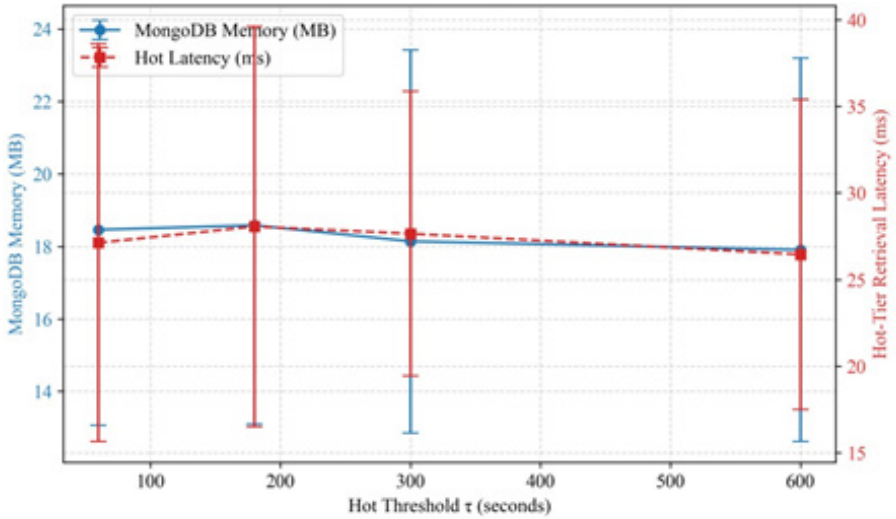

Figure 6 – Experiment 3: Threshold Sensitivity – Memory vs Latency

Experiment 4: Scalability

Experiment 4 evaluates ILM behavior across ingestion rates of 200, 500, and 1,000 msg/s over 20-minute runs with = 300 s. MongoDB memory remained constant at about 20 MB at all three rates, as seen in Table 6 and Figure 7. Flink throughput increased with ingestion rate (444–458 rps) while p95 latency stayed at 1.0 ms. The decreased standard deviation in MongoDB memory at increasing rates (±1.29 → ±0.42 MB) indicates that the hot-tier working set is more reliably stabilized under heavier loads when sweeping occurs more frequently.

Table 6 – Experiment 4: Scalability results

Rate (msg/s)	MongoDB Memory (MB)	Flink Throughput (rps)	Latency p95 (ms)
200	20.62 ± 1.29	444 ± 0.2	1.0
500	21.00 ± 0.86	452 ± 11.1	1.0
1000	20.75 ± 0.42	458 ± 2.2	1.0

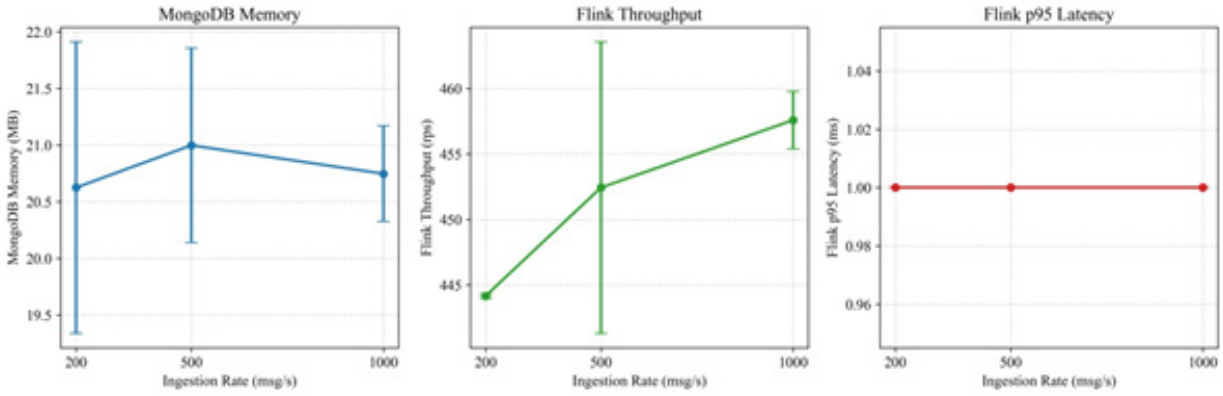

Figure 7 – Experiment 4: scalability across ingestion rates

Experiment 1_1: Extended 90-Minute Validation

Experiment 1_1 was a single 90-minute run at 300 msg/s with = 300 s and a reduced = 2,700 s (45 minutes), designed to activate all three storage tiers within the run duration.

The two-panel result is shown in Fig. 8. For the duration of the 90-minute run, MongoDB hot-tier memory remained bounded at approximately 20 MB. Records were accumulated linearly by TimescaleDB until t ≈ 60 minutes, at which point initiated cold migration. As records started to flow into Parquet, the warm tier stabilized. At t ≈ 60 minutes, the first Parquet file appeared, and at the end of the run, there were two files. Over the course of the run, TimescaleDB received 2.28 million warm records. These findings validate proper end-to-end three-tier lifecycle functioning under continuous, prolonged ingestion.

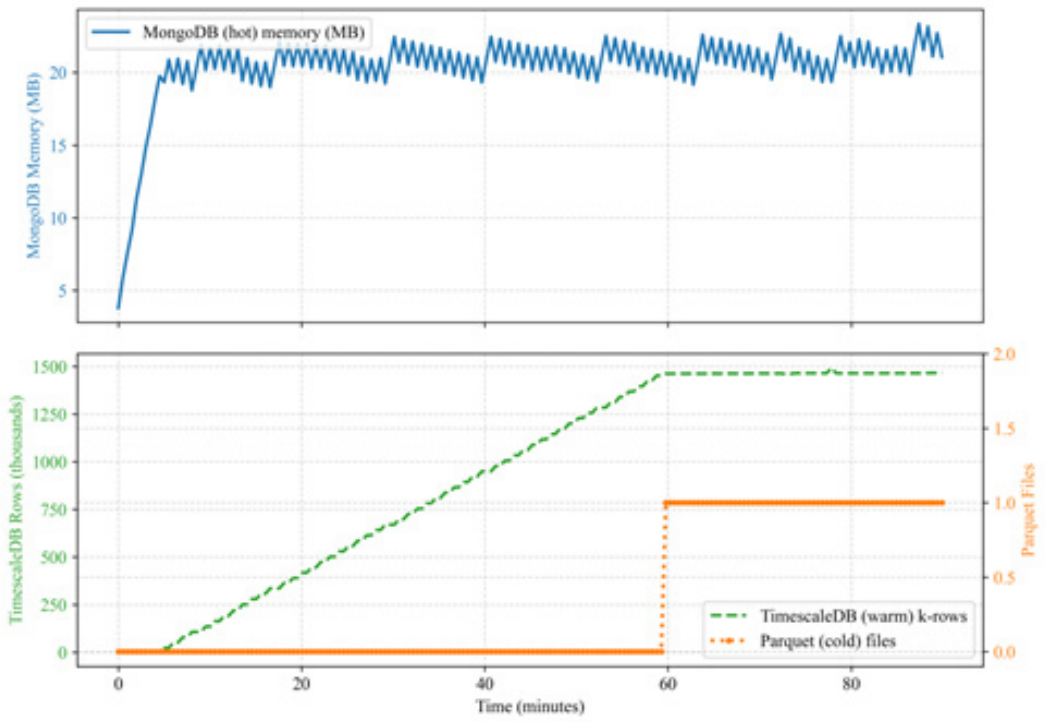

Figure 8 – Experiment 1_1: 90-minute extended run

All five experiments' findings indicate that the proposed ILM system maintains stable stream processing performance, correctly controls the entire three-tier data lifecycle under both short and extended workloads, and achieves stable bounded hot-tier memory regardless of ingestion rate or threshold configuration.

Conclusion

This study presents and evaluates a lightweight, policy-driven Information Lifecycle Management mechanism for containerized real-time streaming pipelines built on Apache Kafka and Apache Flink with a three-tier storage architecture comprising MongoDB, TimescaleDB, and Parquet. The experimental results demonstrate that ILM reduces MongoDB hot-tier peak memory usage by 81% – from 106.96 ± 1.63 MB to 20.28 ± 1.81 MB – with a statistically highly significant effect ($p = 0.000058$) and no measurable impact on Flink throughput or processing latency. Memory bounds remain stable across ingestion rates from 200 to 1,000 messages per second, confirming the scalability of the approach. An extended 90-minute run validates correct three-tier lifecycle operation, with TimescaleDB absorbing 2.28 million warm records and Parquet successfully receiving cold archives.

The system achieves these results using only native database operations and file system primitives, requiring no external orchestration frameworks or specialized hardware, making it immediately deployable in resource-constrained containerized environments.

This study has several limitations. Experiments were conducted on a single laptop rather than a distributed cluster, and performance characteristics may differ at larger deployment scales. The synthetic workload uses a fixed schema with uniform record sizes, which may not reflect real-world data variability. Additionally, the sweeper interval is fixed at 60 seconds rather than dynamically adapted to ingestion rate.

Future work will investigate adaptive threshold scheduling driven by real-time ingestion rate feedback, integration with cloud object storage (Amazon S3, Google Cloud Storage) as the cold tier, and machine learning techniques for predicting optimal record lifetimes from historical access patterns.

REFERENCES

1 Ding, Y. Research on management and optimization of big data computing engine based on cloud native technology IT architecture. Procedia Computer Science, 243, 910–917 (2024). https://doi.org/10.1016/j.procs.2024.09.109

2 Statista. Volume of data/information created, captured, copied, and consumed worldwide from 2010 to 2025. Statista Research Department (2024). URL: https://www.statista.com/statistics/871513/worldwide-data-created/ (accessed 2026.04.10)

3 Sharma, C., and Vaid, A. Leveraging SAP information lifecycle management (ILM): Latest insights and applications. Zenodo, 5(6), 167–173 (2024).

4 Gandomi, A., and Haider, M. Beyond the hype: Big data concepts, methods, and analytics. International Journal of Information Management, 35, 137–144 (2015).

5 Govil, J., Kaur, N., Kaur, H., and Govil, J. Data/information lifecycle management: A solution for taming data beast. Proceedings of the International Conference on Information Technology: New Generations (ITNG 2008), 1226–1227 (2008).

6 Salutina, T.Y., Klesareva, E.Y., and Platunina, G.P. Big data as a management decision-making tool in digital business environments. Proceedings of the 2021 IEEE International Conference on Quality Management, Transport and Information Security, Information Technologies (IT&QM&IS 2021), 893–895 (2021).

7 Ahmad, P.H., and Rai, M. Analysis of optimization strategies for big data storage management: A study. Proceedings of the 2023 4th International Conference on Electronics and Sustainable Communication Systems (ICESC 2023), 1747–1753 (2023).

8 Wang, M.F., Lin, W.T., Tang, C.H., and Tsai, M.F. Constructing storage capacity migration policies for information lifecycle management system. Proceedings of the 12th IEEE International Conference on High Performance Computing and Communications (HPCC 2010), 503–508 (2010).

9 Li-Zhen, S. Research on hierarchical storage of digital library based on the information lifecycle management. Proceedings of the 2010 2nd IEEE International Conference on Information Management and Engineering, 64–66 (2010).

10 HongJu, X., Fei, W., FenMei, W., and XiuZhen, W. Some key problems of data management in army data engineering based on big data. Proceedings of the 2017 IEEE 2nd International Conference on Big Data Analysis (ICBDA 2017), 149–152 (2017).

11 Miao, T. The optimization of land resource data management model in big data era. Proceedings of the IEEE Advanced Information Technology, Electronic and Automation Control Conference (IAEAC 2024), 763–767 (2024).

12 Muratov, S.Y., and Muravyov, S.B. Framework architecture of a secure big data lake. Procedia Computer Science, 229, 39–46 (2023).

13 Kreps, J., Narkhede, N., and Rao, J. Kafka: A distributed messaging system for log processing. Proceedings of the NetDB Workshop, 11, 1–7 (2011).

14 Carbone, P., Katsifodimos, A., Ewen, S., Markl, V., Haridi, S., and Tzoumas, K. Apache Flink: Stream and batch processing in a single engine. Bulletin of the Technical Committee on Data Engineering, 38(4) (2015).

15 Chintapalli, S., Dagit, D., Evans, B., Farivar, R., Graves, T., Holderbaum, M., and Poulosky, P. Benchmarking streaming computation engines: Storm, Flink and Spark Streaming. Proceedings of the 2016 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW 2016), 1789–1792 (2016).

16 Kulkarni, A., and Freedman, M. TimescaleDB: SQL made scalable for time-series data. Proceedings of the CIDR Workshop (2017). URL: https://api.semanticscholar.org/CorpusID:34446750

17 Apache Parquet Project. Apache Parquet format specification (2024). URL: https://parquet.apache.org/docs/ (accessed: 2026.04.10).

[1]Оңаева А.Е.,
магистр, ORCID ID: 0009-0004-4379-1065,
e-mail: al_onayeva@kbtu.kz

[1]Қазақстан-Британ техникалық университеті, Алматы қ., Қазақстан

KAFKA–FLINK КОНТЕЙНЕРЛЕНГЕН АҒЫНДЫҚ ЖҮЙЕЛЕРІНДЕ HOT-TIER ЖАДЫН БАСҚАРУҒА АРНАЛҒАН ҮШ ДЕҢГЕЙЛІ АҚПАРАТ ӨМІРЛІК ЦИКЛЫН БАСҚАРУ МЕХАНИЗМІ

Аңдатпа

Нақты уақыттағы ағындық жүйелер үздіксіз деректер ағынының әсерінен жадқа тұрақты жүктемеге ұшырайды, бұл hot-tier сақтау қабатының шексіз өсуіне әкеледі. Ақпараттың өмірлік циклін басқару (Information Lifecycle Management, ILM) тәсілдері негізінен корпоративтік архивтеу жүйелерінде қолданылған және көпдеңгейлі жад архитектурасын пайдаланатын контейнерленген ағындық пайплайндарда жеткілікті деңгейде зерттелмеген. Бұл жұмыста саясатқа негізделген жеңіл ILM механизмі ұсынылады, ол Kafka–Apache Flink пайплайнына енгізілген және үш деңгейлі сақтау моделін қамтиды: MongoDB (hot-tier), TimescaleDB (warm-tier) және Parquet файлдары (cold-tier). Асинхронды тазалаушы ағын (sweeper thread) жазбаларды τ_hot және τ_cold уақыт шектеріне сәйкес деңгейлер арасында жылжытады, осылайша ағындық өңдеуді бұзбай, hot-tier толып кетуін болдырмайды. Жад тиімділігін, деңгейлер бойынша деректерге қол жеткізу кідірісін, шектерге сезімталдықты, масштабталуды және ұзақ мерзімді жұмыс тәртібін бағалау үшін бес эксперимент жүргізілді. Нәтижелер ILM қолдану MongoDB жадын ең жоғары деңгейде пайдалануды 81%-ға азайтатынын көрсетті (106.96 ± 1.63 МБ-тан 20.28 ± 1.81 МБ-қа дейін, $p < 0.001$), бұл ретте Flink жүйесінің өткізу қабілеті мен өңдеу кідірісі өзгеріссіз қалады. Жад шектеулері секундына 200-ден 1000-ға дейін хабарлама жылдамдығында тұрақты сақталады. 90 минуттық қосымша эксперимент үш деңгейлі өмірлік циклдің дұрыс жұмыс істейтінін дәлелдейді: MongoDB тұрақты шекте қалады, TimescaleDB 2.28

миллион warm-tier жазбасын жинақтайды, ал Parquet cold-tier архивін қалыптастырады. Бұл нәтижелер нақты уақыттағы контейнерленген ағындық жүйелерде деректердің өмірлік циклін тиімді және аз шығынмен басқару тек дерекқорлардың ішкі мүмкіндіктері мен файлдық жүйе операцияларын пайдалану арқылы жүзеге асырылатынын көрсетеді.

Түйін сөздер: ақпараттың өмірлік циклін басқару (ILM), Apache Kafka, Apache Flink, MongoDB, TimescaleDB, нақты уақыттағы ағындық өңдеу, жадты оңтайландыру, көпдеңгейлі сақтау, контейнерленген пайплайндар, деректерді архивтеу.

[1]Онаева А.Е.,
магистр, ORCID ID: 0009-0004-4379-1065,
e-mail: al_onayeva@kbtu.kz

[1]Казахстанско-Британский технический университет, г. Алматы, Казахстан

ТРЕХУРОВНЕВЫЙ МЕХАНИЗМ УПРАВЛЕНИЯ ЖИЗНЕННЫМ ЦИКЛОМ ИНФОРМАЦИИ ДЛЯ КОНТРОЛЯ ПАМЯТИ ГОРЯЧЕГО УРОВНЯ В КОНТЕЙНЕРИЗОВАННЫХ СТРИМИНГ-ПАЙПЛАЙНАХ KAFKA–FLINK

Аннотация

Системы потоковой обработки в реальном времени сталкиваются с постоянной нагрузкой на память, поскольку непрерывное поступление данных приводит к неограниченному росту хранилища горячего уровня. Существующие подходы управления жизненным циклом информации (Information Lifecycle Management, ILM) в основном применялись в корпоративных системах архивирования и не были исследованы в контексте контейнеризованных стриминговых пайплайнов, использующих многоуровневые архитектуры хранения в памяти. В данной работе представлен легкий механизм ILM, управляемый политиками и интегрированный в пайплайн Kafka–Apache Flink с трехуровневой моделью хранения: MongoDB (горячий уровень), TimescaleDB (теплый уровень) и файлы Parquet (холодный уровень). Асинхронный поток очистки (sweeper thread) перемещает записи между уровнями в соответствии с настраиваемыми временными порогами и , предотвращая переполнение горячего уровня без нарушения потоковой обработки. Было проведено пять экспериментов для оценки эффективности использования памяти, задержки доступа к данным по уровням, чувствительности к пороговым значениям, масштабируемости и поведения при длительной эксплуатации. Результаты показывают, что применение ILM снижает пиковое использование памяти MongoDB на 81% (с 106.96 ± 1.63 МБ до 20.28 ± 1.81 МБ, $p < 0.001$), при этом пропускная способность и задержка обработки в Flink остаются неизменными. Ограничения по памяти стабильно соблюдаются при скорости поступления данных от 200 до 1000 сообщений в секунду. Дополнительный эксперимент продолжительностью 90 минут подтверждает корректную работу трехуровневого жизненного цикла: MongoDB остается в заданных пределах, TimescaleDB аккумулирует 2.28 миллиона записей теплого уровня, а Parquet формирует холодный архив. Полученные результаты подтверждают, что эффективное и малозатратное управление жизненным циклом данных возможно в контейнеризованных системах потоковой обработки в реальном времени с использованием только встроенных возможностей баз данных и операций файловой системы.

Ключевые слова: управление жизненным циклом информации (ILM), Apache Kafka, Apache Flink, MongoDB, TimescaleDB, потоковая обработка в реальном времени, оптимизация памяти, многоуровневое хранение, контейнеризованные пайплайны, архивирование данных.