

Vyacheslav Lemeshko

Postgraduate student of the Faculty of Informatics and Computer Engineering, Igor Sikorsky Kyiv Polytechnic Institute
National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute"

ORCID ID: 0009-0008-1495-715X

slavalemeshko@gmail.com

Dmytro Boyko

Postgraduate student of the Department of Information Systems and Technologies
Taras Shevchenko National University of Kyiv, Kyiv, Ukraine

ORCID ID: 0009-0006-1195-1065

boyko.dmytro@knu.ua

Ihor Parkhomey

Doctor of Technical Sciences, Professo, Professor Department of Information Systems and Technologies
Taras Shevchenko National University of Kyiv, Kyiv, Ukraine

ORCID: 0000-0002-9510-7657

ihor.parhomey@knu.ua

DATA UPDATE ALGORITHM BASED ON TIME CRITERIA

Abstract. This paper presents an enhanced approach to logging systems for business services by implementing a modified producer-consumer pattern with time-based data aggregation. Traditional logging approaches that register each business process step using individual database queries places a significant load on relational database management systems (DBMS), resulting in performance bottlenecks. The proposed solution introduces a "smart" buffer that operates in accumulation mode, aggregating log messages over a defined time interval (T_{merge}) before batch processing them to the database, reducing the frequency of database operations.

The mathematical model extends the classic producer-consumer pattern by incorporating an aggregation interval that governs when accumulated data are processed. This ensures optimal buffer utilization while minimizing database transactions. The model defines stability conditions that balance production and consumption rates to prevent buffer overflow or underflow scenarios, adapting to varying loads.

Empirical testing on a web server (4 cores, 16GB RAM) and SQL database server (4 cores, 32GB RAM) demonstrated significant improvements in system performance. Analysis of Microsoft SQL Server's Query Store showed reduced query execution frequency, decreased duration, and lower logical write operations. While CPU usage remained stable, there was a minor increase in RAM usage due to batch storage requirements. These results confirm that the accumulation-based buffer effectively reduces database load without compromising logging fidelity.

The approach aligns with recent research trends in streaming data processing, which increasingly favor batch processing and adaptive interval mechanisms. The solution offers flexibility through configurable parameters like the T_{merge} interval and buffer capacity. Future research directions include developing real-time adaptive aggregation mechanisms and extending the model to distributed multi-node environments.

Keywords: producer-consumer pattern, logging systems model, time-based aggregation, buffer optimization, batch processing, mathematical model, query optimization, asynchronous processing.

Лемешко В'ячеслав Анатолійович

аспірант факультету інформатики та обчислювальної техніки КПІ ім.Ігоря Сікорського

Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського»

ORCID ID: 0009-0008-1495-715X

slavalemeshko@gmail.com

Бойко Дмитро Михайлович

Аспірант кафедри інформаційних систем та технологій

Київський національний університет імені Тараса Шевченка, м.Київ, Україна

ORCID ID: 0009-0006-1195-1065

boyko.dmytro@knu.ua

Пархомей Ігор Ростиславович

Д.т.н., професор, професор кафедри інформаційних систем та технологій

Київський національний університет імені Тараса Шевченка, м.Київ, Україна

ORCID ID: 0000-0002-9510-7657

ihor.parhomey@knu.ua

© 2026 Lemeshko V., Boyko D., Parkhomey I. This work is licensed under CC BY 4.0.

<https://creativecommons.org/licenses/by/4.0/>

АЛГОРИТМ ОНОВЛЕННЯ ДАНИХ НА ОСНОВІ КРИТЕРІЇВ ЧАСУ

Анотація. У цій статті представлено вдосконалений підхід до систем ведення журналу для бізнес-послуг шляхом реалізації модифікованого шаблону «виробник-споживач» з агрегацією даних на основі часу. Традиційні підходи до ведення журналу, які реєструють кожен крок бізнес-процесу за допомогою окремих запитів до бази даних, створюють значне навантаження на реляційні системи керування базами даних (СКБД), що призводить до вузьких місць у продуктивності. Запропоноване рішення впроваджує «розумний» буфер, який працює в режимі накопичення, агрегуючи повідомлення журналу протягом визначеного інтервалу часу (T_{merge}) перед пакетною обробкою їх у базі даних, зменшуючи частоту операцій з базою даних.

Математична модель розширює класичний шаблон «виробник-споживач», включаючи інтервал агрегації, який визначає, коли обробляються накопичені дані. Це забезпечує оптимальне використання буфера, мінімізуючи транзакції бази даних. Модель визначає умови стабільності, які балансують швидкість виробництва та споживання, щоб запобігти сценаріям переповнення або недоповнення буфера, адаптуючись до змінних навантажень.

Емпіричне тестування на веб-сервері (4 ядра, 16 ГБ оперативної пам'яті) та сервері бази даних SQL (4 ядра, 32 ГБ оперативної пам'яті) продемонструвало значне покращення продуктивності системи. Аналіз сховища запитів Microsoft SQL Server показав зниження частоти виконання запитів, зменшення тривалості та меншу кількість логічних операцій запису. Хоча використання процесора залишалося стабільним, спостерігалось незначне збільшення використання оперативної пам'яті через вимоги до пакетного зберігання. Ці результати підтверджують, що буфер на основі накопичення ефективно зменшує навантаження на базу даних без шкоди для точності реєстрації.

Цей підхід відповідає останнім тенденціям досліджень у потоковій обробці даних, які все більше сприяють пакетній обробці та адаптивним інтервальним механізмам. Рішення пропонує гнучкість завдяки налаштовуваним параметрам, таким як інтервал T_{merge} та ємність буфера. Майбутні напрямки досліджень включають розробку адаптивних механізмів агрегації в реальному часі та розширення моделі на розподілені багатовузлові середовища.

Ключові слова: шаблон виробник-споживач, модель систем реєстрації, агрегація на основі часу, оптимізація буфера, пакетна обробка, математична модель, оптимізація запитів, асинхронна обробка.

1. Introduction

Logs make it possible to quickly identify the cause of failures or errors. They can store information about which stage of a business service encountered a problem, what parameters were passed, and what the result of execution was. Logging allows for tracking system performance, identifying bottlenecks, and optimizing operation. Logs help analyze how users interact with the system, revealing inefficiencies or opportunities for improvement.

To ensure an adequate level of business service execution, it is necessary to continuously monitor the system's state. The implementation and integration of a logging subsystem can pose various challenges, such as large data volumes, elevated impact on system performance, and complexities in data analysis.

2. Statement of the problem

Traditional approaches to business service logging, which involve writing each event to the database separately, result in a significant number of write operations and increased load on the DBMS, especially with intensive log generation. This work focuses on a logging subsystem that stores only the latest, most relevant state of a business service execution, without retaining the detailed historical execution path. For optimization purposes, an enhanced producer-consumer model with time-based aggregation (T_{merge}) is proposed.

3. Analysis of recent studies and publications

A review of recent scientific works [2]-[16] indicates substantial interest in optimizing logging subsystems and processing streaming data, especially in the context of reducing the load on DBMSs and improving scalability. Modern streaming and distributed systems increasingly adopt “batch” or microbatch approaches [12],[13],[16], which reduce the volume of transactions and increase performance. Studies dedicated to the producer-consumer model [3], [8] emphasize the importance of maintaining a balance between data generation speed and data consumption speed, as well as a flexible approach to resource allocation in real-time. The use of adaptive aggregation intervals [6], [10], [16] has proven effective in scenarios with high log volumes and uneven data arrival frequencies, noticeably reducing overall write overhead and increasing resilience to peak loads.

Other studies [4], [5], [7], [9], [11] focus on aspects of implementing distributed and scalable logging systems, proposing methods for structured storage, offloading central nodes, and fast search. In particular, they highlight the importance of optimized buffers (elastic, with dynamic resizing) and algorithms for harmonizing write and read rates.

Thus, most research underscores the need for balanced buffer and aggregation mechanisms to avoid overwhelming the system with frequent single transactions. At the same time, there is no unified model that integrates the classic producer-consumer pattern with a regulated aggregation interval, particularly for relational DBMSs, which are widely used in numerous business services.

4. Research objectives and tasks

The aim of the article is to improve the efficiency of business service logging by developing a modified producer-consumer template with time-based data aggregation.

5. Research results

Logging system model.

Consider the process of logging the steps in a business process. Each step of the business process triggers a data-registration function in an abstraction implemented using the “repository” design pattern. The implementation of

this abstraction sends a query to the relational DBMS to create a new record in relational tables. After the business-service step completes, a data-update function in the “repository” abstraction is called. The implementation of this abstraction sends a query to the relational DBMS to update an existing record in the relational tables (Figure 1).

This approach has several advantages and disadvantages. Its advantages include that logging each step of the business process enables detailed real-time tracking of its execution, aiding in troubleshooting and increasing system transparency. Using the repository pattern provides a clear separation of business logic from database operations, making it easier to modify or enhance system components. Abstraction via the repository simplifies maintenance and testing by allowing changes to the internal data-access implementation without affecting business logic. Using a relational DBMS as the data store is a standard approach, making the solution understandable to many developers and easily integrable with other systems. Logs are stored in a structured format in the relational database, making them convenient for building reports, analyzing process efficiency, and identifying bottlenecks.

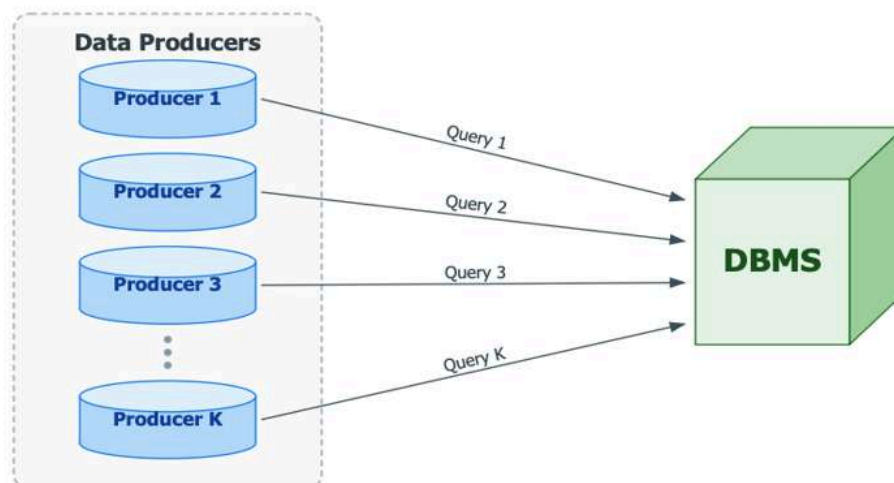


Fig. 1. General diagram of the first approach.

However, frequent write and update operations place a heavy load on the relational DBMS, especially if there is a high degree of parallelism or a large number of business-service steps. Continuous database access for insert and update slows down business-service execution. If each step is treated as a separate transaction, this can lead to delays due to resource locks. Frequent updates and log storage in the database lead to the accumulation of large amounts of historical data, which must be periodically cleaned up.

To reduce the load on the DBMS, minimize the data sent over the network, reduce operations on the disk subsystem, and speed up execution of business-service steps, an improvement has been proposed and implemented. A “smart” buffer has been introduced between the business-service execution subsystem and the logging subsystem, which asynchronously receives logging data in the form of messages and performs a merge (aggregation) operation on these messages before writing to minimize the number of DBMS queries (Figure 2).

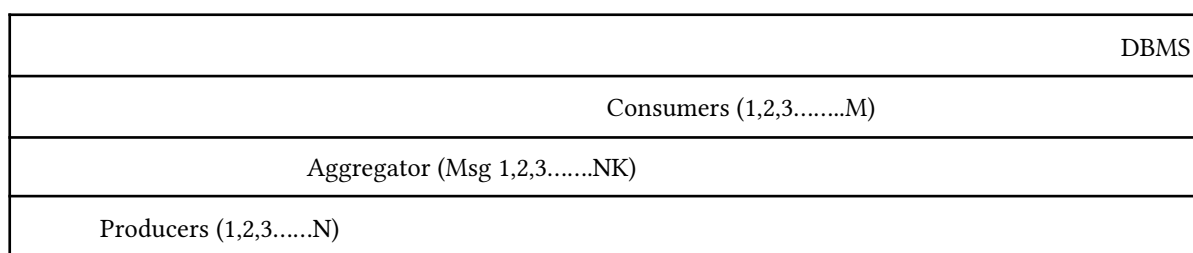


Fig. 2. General diagram of the proposed approach.

By using a feature toggle mechanism, it became possible to introduce the new implementation with an option to switch back to the old one. Under the new approach, each step of a business-service instance generates logging data in the form of messages, which are accumulated in a queue and transformed into database queries by several consumers into queries for the relational DBMS. The logging subsystem follows the “producer-queue-consumer” model [16].

The mathematical model of the producer-consumer pattern describes the interaction between two types of processes: producers, which generate data or “products,” and consumers, which consume these data. They typically share a buffer of limited size for data exchange. This model helps clarify the optimal conditions for matching both sides’ performance and avoiding excessive waiting or blocking.

Model Variables:

N - the size of the shared buffer. This is the maximum number of elements that can be stored simultaneously.

$P(t)$ - the number of elements produced by the producers by time t .

$C(t)$ - the number of elements consumed by the consumers by time t .

$B(t)$ - the number of elements in the buffer at time t . This is the buffer's state variable, updated over time according to producer and consumer actions.

r_p and r_c - the average production (producer) and consumption (consumer) rates, respectively.

$$B(t) = P(t) - C(t), \quad (1)$$

where $B(t)$ is bounded between 0 and N :

$$N \geq B(t) \geq 0. \quad (2)$$

The rate of change in the number of elements in the buffer can be expressed as a derivative:

$$\frac{dB(t)}{dt} = \frac{dP(t)}{dt} - \frac{dC(t)}{dt} = r_p - r_c. \quad (3)$$

Since $dP \frac{(t)}{dt} = r_p$ and $dC \frac{(t)}{dt} = r_c$. This simplifies to

$$\frac{dB(t)}{dt} = r_p - r_c. \quad (4)$$

Stability Conditions:

Buffer overflow condition: If $r_p > r_c$, the buffer will gradually fill, and once $B(t) = N$, producers have to stop or wait, resulting in delays.

Buffer underflow condition: If $r_p < r_c$, the buffer empties, causing consumers to wait for new data. a web server

Stable system operation: If $r_p = r_c$. In this case, the buffer remains in equilibrium without overflowing or being empty.

The mathematical model of the producer-consumer pattern helps determine how to maintain balance between production and consumption speeds. This ensures optimal buffer usage, minimizing delays for both producers and consumers.

However, this approach does not reduce the number of queries to the DBMS. To lower the number of DBMS queries, we propose changing the buffer's behavior: elements are processed at specific intervals (the aggregation interval T_{merge}), meaning consumers work in an accumulation mode. We analyze a model wherein data are consumed in a batch once T_{merge} elapses, which significantly affects the overall load on the DBMS.

Additional model variables:

T_{merge} - the aggregation interval (the period over which data are accumulated before processing).

$M(t)$ - the number of elements accumulated in the buffer during T_{merge} before consumption.

$C(t)$ - the number of consumers simultaneously processing the accumulated data. Item 2

Now the buffer state depends on the following phases:

Accumulation phase. During time T_{merge} , elements are added to the buffer but not consumed.

Consumption phase. After T_{merge} ends, all accumulated elements are consumed by the $C(t)$ consumers in one request.

The equation for the buffer state $B(t)$ remains as in (1):

$$c(t) = \{0, t \bmod T_{merge} \neq 0(B(t), C(t) \cdot M(t)), t \bmod T_{merge} = 0 \quad (5)$$

Accumulation phase ($t \bmod T_{merge} \neq 0$):

Consumption does not occur $C(t) = 0$;

The buffer grows at the rate set by producers r_p .

Consumption phase ($t \bmod T_{merge} = 0$):

All $C(t)$ consumers simultaneously consume the data.

The maximum number of data elements that can be consumed is: $C(t) * M(t)$, where

$$M(t) = \int_{t-T_{merge}}^t p(t)dt, \quad (6)$$

number of elements accumulated over T_{merge} .

If $B(t) < C(t) * M(t)$, the entire buffer is consumed, leaving $B(t) = 0$.

To prevent buffer overflow, the average consumption rate over T_{merge} must match the production rate:

$$C(t) * M(t) \geq r_p * T_{merge}. \quad (7)$$

Analysis of the results of the proposed approach.

Experiments were conducted on the following computing resources: a web server (4 cores, 16 GB RAM) and an SQL database server (4 cores, 32 GB RAM). To evaluate the effect on the Microsoft SQL Server database server, we analyzed the following metrics: Execution Count (number of query executions), Total Duration (ms) (total query execution time in milliseconds), and Total Logical Writes (KB) (total volume of logical writes in kilobytes). The choice of these metrics is justified by the need for relevant analysis, minimizing external factors, and focusing on aspects most significant for MS SQL Server performance under the studied conditions.

From the CPU usage graphs for the web server before (Figure 3a) and after (Figure 3b) the logging subsystem implementation, it can be concluded that the overall CPU load remains similar. Both charts show peaks in CPU usage with approximately the same frequency and amplitude. Although after the changes (Figure 3b) the spikes in load are somewhat less pronounced, the general trend is preserved. This indicates that introducing the logging subsystem does not significantly affect CPU load - a fairly positive outcome.

In the graph before the changes (Figure 4a), RAM usage is relatively stable with periodic peak loads. After implementing the logging subsystem (Figure 4b), the overall load pattern is maintained, but there is a slight increase in the baseline level of memory usage. This may indicate increased RAM consumption due to the new features. Meanwhile, peak load frequency and intensity remain similar, implying that the main usage patterns of the web server are unchanged.

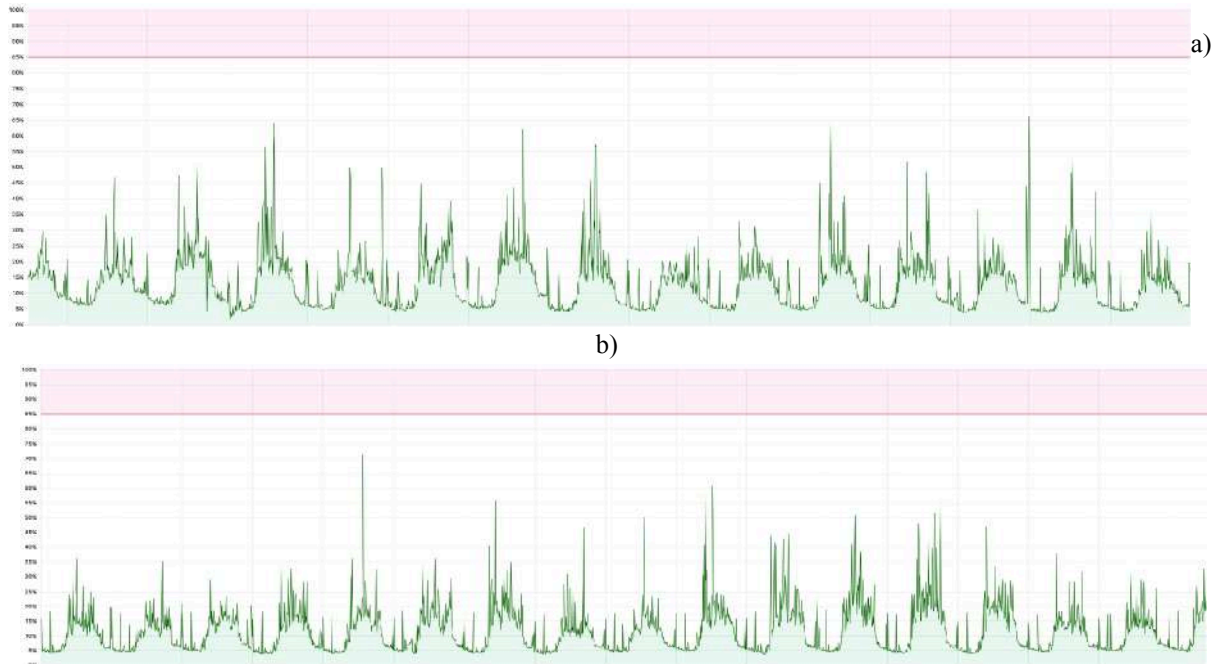


Fig.3(a) and 3(b). CPU resource usage of the web server.



4(a) and

4(b). RAM resource usage of the web server.

The most significant change is the disappearance in Query Store of the query with ID 121041091 (Figure 5a), which, prior to the logging implementation, had the highest call frequency (shown in green). It updated the log data. This indicates that its execution became much less frequent, thereby reducing its impact on the DB server. From the graph after the changes (Figure 5b), one can see that other queries had a redistribution of call frequency - for instance, the most frequent queries with IDs 121041054, 121041093, and 121041092 show an increase in the number of calls, reflecting the system's adaptation to the new logging approach.

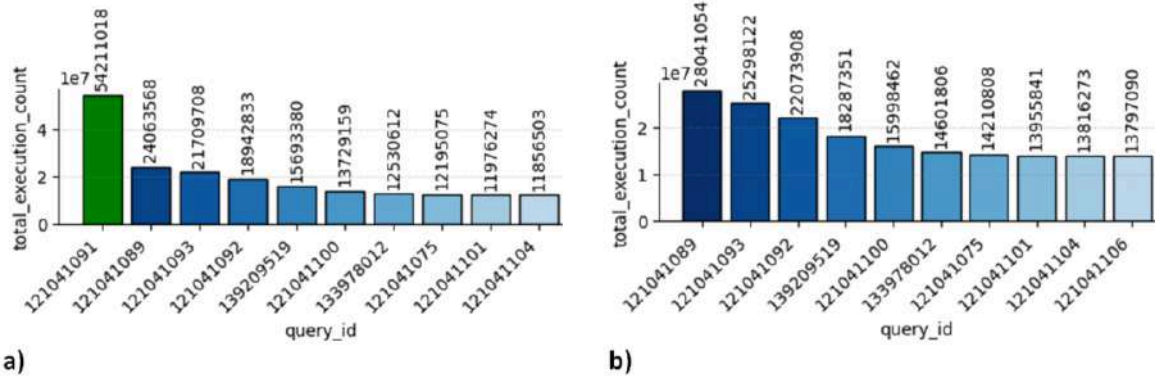


Fig. 5(a) and 5(b). Total query calls.

Before implementing the proposed logging subsystem approach (Figure 6a), the query with ID 121041091 (highlighted in green) contributed significantly to the total query duration. After the changes (Figure 6b), this query no longer appears in Query Store, indicating its elimination or substantially decreased execution frequency. This significantly reduced the load on the database server. Some queries with previously high total execution times (e.g., 121041912) either no longer appear or exhibit noticeably lower execution times. This points to optimization or reduced frequency of their use.

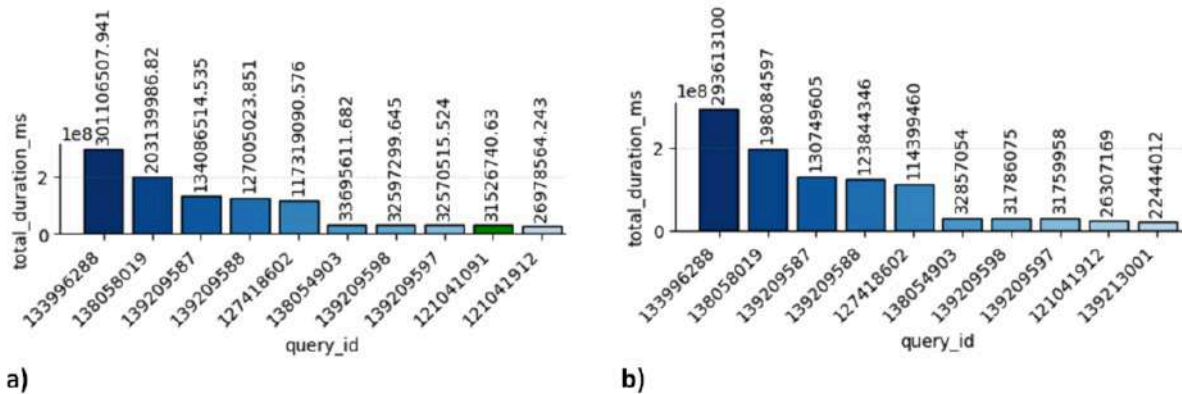


Fig. 6(a) and 6(b). Total query duration (ms).

Comparing both graphs (Figures 7a and 7b), we see an overall reduction in the number of logical writes after implementing the logging subsystem. On graph 7b, the peak values of total logical writes are substantially lower than on graph 7a, indicating improved efficiency of query processing. Reducing the overall number of logical writes without a significant increase in the frequency of small queries suggests improved write algorithms and likely fewer redundant or duplicate logging operations. Hence, batch processing enabled by the accumulation-based mode reduced the number of database calls and provided more stable load distribution.

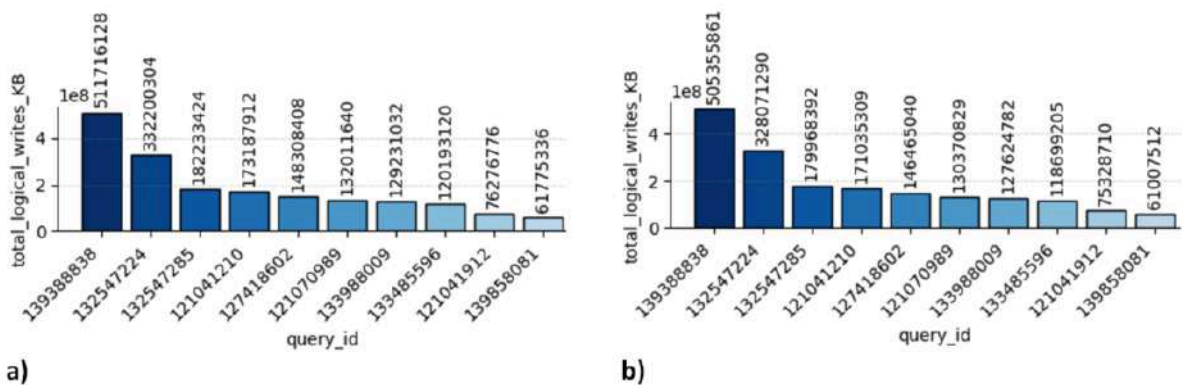


Fig. 7(a) and 7(b). Total logical writes.

6. Conclusions and prospects for further research

This work examined and analyzed an approach to business-service logging based on the Producer-Consumer model with a buffer that operates in accumulation mode and periodically (in batch) processes the data. The study confirmed that the Producer-Consumer mathematical model incorporating an aggregation interval T_{merge} reduces the number of queries to the DBMS, because logs are recorded in groups instead of through continuous single write operations. Empirical results show a reduction in the total number of logical writes, a decrease in database server CPU load, and a slight increase in the web server's RAM usage (due to storing the batch in memory) [16].

A review of recent scientific research on buffering and logging shows that worldwide practice is likewise moving to batch message processing and adaptive periodic intervals. This confirms the promise of the proposed approach and aligns with the trend of optimizing logging to reduce overhead and improve system resilience.

The implemented solution significantly reduces the number of write transactions to the database, while still providing sufficient log detail for further analysis.

The proposed algorithm can be adapted to a range of conditions: varying log generation rates, different numbers of consumers, and specific performance requirements. With proper configuration of the T_{merge} period and buffer capacity, the approach ensures stable system performance even under intensive log generation. Future research may focus on enhancing real-time adaptive aggregation interval mechanisms (based on prior load analysis and forecasting) and extending the model to distributed environments in which logging is performed simultaneously on multiple nodes.

Authors' contributions.

Vyacheslav Lemeshko – development of the proposed approach, simulation modeling, participation in the formation of the research conclusions. Dmytro Boyko – preparation of the literature review, calculation of model parameters, preparation of graphs, participation in the formation of the research results. Ihor Parkhomey – determiner of the scientific and practical task, participation in the formation of the research conclusions.

Declaration on artificial intelligence. The authors did not use artificial intelligence when creating the materials of the article.

Conflict of interest. The author declares that there is no conflict of interest and confirms that during the preparation of this work there were no commercial, financial, or other relationships that could be construed as influencing the results of the study or their interpretation. The work was performed in accordance with the principles of academic integrity, ethical standards for conducting scientific research, and editorial policy requirements for preventing conflicts of interest.

References

1. Monitoring performance by using the Query Store. Microsoft Docs. URL: <https://learn.microsoft.com/en-us/sql/relational-databases/performance/monitoring-performance-by-using-the-query-store?view=sql-server-ver16>.
2. Sax, M. J. (2020). Performance optimizations and operator semantics for streaming data flow programs (Doctoral dissertation, Mathematisch-Naturwissenschaftliche Fakultät). DOI:10.18452/21424
3. Sharma, M. K. (2020) A Design Space for Distributed Producer-Consumer Data Structures Using RDMA, Master's thesis, University of Waterloo, Waterloo, Canada. URL: https://uwaterloo.ca/bitstream/10012/16163/4/Sharma_Manoj.pdf.
4. Foyer, C. M. (2021) Abstractions for Portable Data Management in Heterogeneous Memory Systems, PhD thesis, University of Bristol, Bristol, United Kingdom. URL: <https://research-information.bris.ac.uk/ws/portalfiles/portal/281444302>.
5. Marcu, O.C., Costan, A., et al, (2018) Storage and Ingestion Systems in Support of Stream Processing: A Survey. [Technical Report] RT-0501, INRIA Rennes - Bretagne Atlantique and University of Rennes 1, France. pp.1-33. URL: <https://inria.hal.science/hal-01939280/file/RT-0501v2.pdf>
6. Shah, M. A., Hellerstein, J. M. (2002) Flux: An adaptive partitioning operator for continuous query systems. Report No. UCB/CSD-2-1205. Computer Science Division (EECS), University of California, Berkeley, California, pp.1-16 URL: <https://www2.eecs.berkeley.edu/Pubs/TechRpts/2002/CSD-02-1205.pdf>.
7. John, V., Liu, X. (2017) A survey of distributed message broker queues. arXiv preprint arXiv:1704.00411, pp.1-8. URL: <https://arxiv.org/abs/1704.00411>.
8. Lohrmann, B., Janacik, P., Kao, O. (2015) Elastic stream processing with latency guarantees. 2015 IEEE 35th International Conference on Distributed Computing Systems (ICDCS), Columbus, OH, USA, June 29 - July 2, 2015, pp.1-12. URL: https://web.archive.org/web/20170809095959/http://www.cit.tu-berlin.de/fileadmin/a34331500/misc/icdcs15_preprint.pdf.
9. Marcu, O.C., Costan, A., Antoniu, G., Pérez-Hernández, M. S. (2017) Kera: A Unified Storage and Ingestion Architecture for Efficient Stream Processing, Research Report RR-9074, INRIA Rennes - Bretagne Atlantique, pp.1-28. URL: <https://inria.hal.science/hal-01532070/document>.
10. Parrinello, A. (2021) Benchmarking materialized views of SQL-based stream processing systems, Master's thesis, Università di Bologna, Bologna, Italia, URL: <https://amslaurea.unibo.it/id/eprint/30885/1/thesis.pdf>.
11. Doan, Q. T. (2021) A Framework for Integrating IoT Streaming Data from Multiple Sources, PhD thesis, La Trobe University, Melbourne, Victoria, Australia, <https://doi.org/10.26181/17211713.v1>.
12. Zhang, H., Dong, H., Li, G. & Zhang, C. (2024) Elastic buffer-based logging for high-throughput cloud services. Proceedings of the VLDB Endowment 17 3869–3882. <https://doi.org/10.14778/3685800.3685812>.
13. Lee, S., Jeong, Y., Park, K. et al (2023), zStream: towards a low latency micro-batch streaming system. Cluster Comput 26, 2773–2787. <https://doi.org/10.1007/s10586-022-03758-1>.
14. Bharti, U., Bajaj, D., Goel, A., Gupta, S. C. (2019) Identifying requirements for Big Data analytics and mapping to Hadoop tools. International Journal of Recent Technology and Engineering 8 4384-4392. <https://doi.org/10.35940/ijrte.C5524.098319>.
15. Lee, S., Jeong, Y., Kim, M., Park, S. (2021) Q-Spark: QoS Aware Micro-batch Stream Processing System Using Spark, 2021 IEEE International Conference on Autonomic Computing and Self-Organizing Systems Companion (ACSOS-C), DC, USA, pp. 38-43, <https://doi.org/10.1109/ACSOS-C52956.2021.00027>.
16. Пархомей, І., Бойко Ю., Лемешко В., Срьоменко О. (2025) Оптимізація логуювання в інформаційних системах на основі математичної моделі управління буфером у шаблоні producer-consumer Міжнародний науково-технічний журнал «Вимірювальна та обчислювальна техніка в технологічних процесах», №2, 2025, с.218-225. <https://doi.org/10.31891/2219-9365-2025-82-30>

Надійшла до редакції: 14.06.2026

Прийнята до друку: 15.09.2026

Опубліковано: 30.09.2026